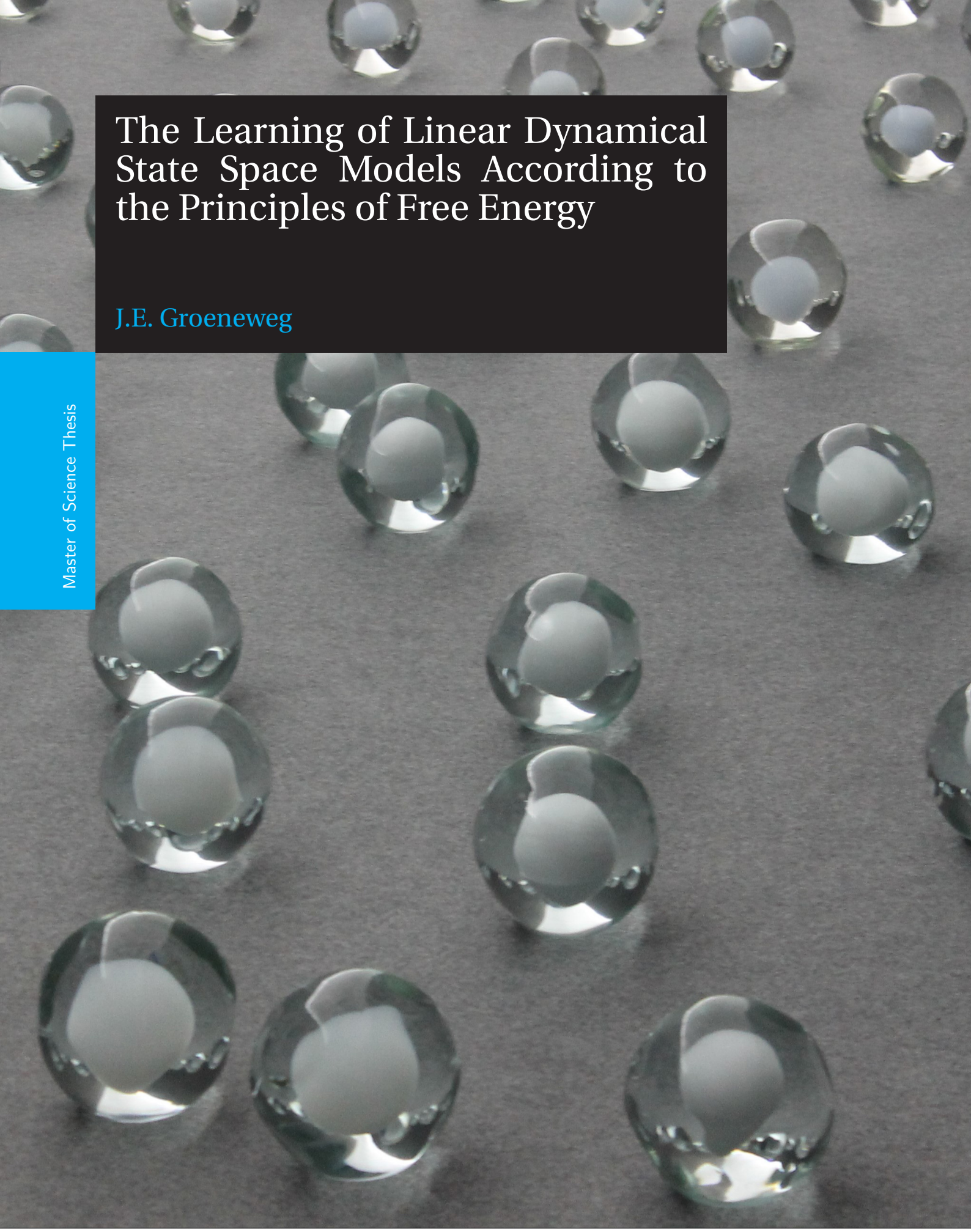


The Learning of Linear Dynamical State Space Models According to the Principles of Free Energy

J.E. Groeneweg

Master of Science Thesis

TU Delft Delft University of Technology

The Learning of Linear Dynamical State Space Models According to the Principles of Free Energy

The improvement of acceptance and utilization of the Machine Learning technique: Expectation Maximization in Control theory

Master of Science Thesis

For the degree of Master of Science in
Mechanical Engineering and System & Control
at Delft University of Technology.

Julia Elina Groeneweg

November 14, 2018

Under the supervision of **Prof. dr. ir. M. Wisse** and **Dr. ir. T.J.J. van den Boom**

Faculty of Mechanical, Maritime and Materials Engineering (3mE)
Delft University of Technology





Copyright ©
All rights reserved.

DELFT UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF
DELFT CENTER FOR SYSTEMS AND CONTROL (DCSC)
AND BIOMECHANICAL DESIGN

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

THE LEARNING OF LINEAR DYNAMICAL STATE SPACE MODELS ACCORDING TO THE PRINCIPLES OF FREE ENERGY

by

JULIA ELINA GROENEWEG

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN
MECHANICAL ENGINEERING
SYSTEM & CONTROL

Dated: November 28, 2018

Supervisor(s):

Prof. dr. ir. M. Wisse

Dr. ir. T.J.J. van den Boom

Reader(s):

Dr. M. Kok

Ir. A. Anil Meera

ABSTRACT

The Free Energy principle represents a Neuroscience theory that unlike any other theory can explain all behavioral aspects of adaptive agents (like humans). The issue of behavior is made computationally tractable by considering behavioral processes as various optimization problems with a single objective: minimization of Free Energy. The rationale is that to exist adaptive agents have to resist a natural tendency to disorder. This is accomplished by adopting a policy that minimizes Surprisal. To put it differently the adaptive agent has to become an expert at predicting its own sensations (eyes and proprioception) caused by an uncertain environment. The brain is therefore believed to embody a generative model. Surprisal is the measure of how good this generative model can explain the sensations experienced by adaptive agent. The mathematical equivalent of Surprisal is the negative log Likelihood. The Likelihood distribution -a typical term used in probability theory- expresses how probable the observed sensory data -sensations- is for different settings of causes. By maximization of the Likelihood Surprisal is minimized. To obtain an exact maximum Likelihood solution one however has to integrate over all possible settings of all known and unknown causes. Rather than minimization of Surprisal a minimization of a bound on Surprisal: the Free Energy is considered. The Free Energy principle formulated into a Bio-inspired Control algorithm could be the solution to the lack of adaptability and precision towards uncertain and unknown environments that the robotic community currently faces. This formulation is however not trivial. This thesis addresses the process of Learning a model of a Process according to the principles of Free Energy. It is identified the Free Energy principle to heavenly rely upon concepts from the Machine Learning methodology Expectation Maximization. In case of linearity and a Wiener process assumption on the noise affecting the Process, the Free Energy Learning algorithm is equivalent to the EM algorithm. An evaluation of the minimization of Free Energy problem with respect to the state led to a state estimation computation equivalent to the Kalman filter. An evaluation of the minimization of the Free Energy problem with respect to the model parameters led to a model parameter computation equivalent to the solution of a linear least square problem. It has been identified the EM algorithm to be quite unfamiliar in Control settings. Implementations of the Free Energy Learning algorithm = EM algorithm yielded promising results although only a naive version has been implemented. Improvements on the implementations are believed to generate a competitive algorithm for solving control problems. Next steps are consideration of the full Free Energy principle algorithm where the Wiener assumption on noise does not apply and the input is derived with a similar Free Energy minimization problem. The former could relate to the concept of incorporating a prediction horizon suggesting a intimate relationship with the Subspace Identification method. The latter is inherently different from conventional control. The full Free Energy principle algorithm does distinguishes itself from the EM algorithm.

Keywords *The Free Energy Principle, Variational Free Energy, Bayesian Inference, Bayesian framework, variational Expectation Maximization, Learning, Maximum Likelihood estimates*

PREFACE

This thesis is the product of a hunch prediction error minimization to explain aspects of human behavior and this might be the reason why humans are so adaptive to unexpected environmental encounters. In contrast to complex robotic application -for example humanoid robots- that resolve in complicated calculations once for example the foot is no longer placed straight on the ground as a consequence of imperfections in the floor. The robotic application attempts to understand the dynamics of the wrong placing of the foot but fails due to the complexity and time constraints of the problem resulting in a large prediction error. Could it be that humans in this sort of encounters rather than trying to understand just control their foot back to the straight position? It seems likely and is also observed in robotic control where Engineers manually place a local controller that makes sure the foot is kept straight. Essentially these controllers are driven by prediction error minimization. This led to the first Google search I performed trying to discover a scientific foundation for the hunch and whether or not an automated control design existed driven by prediction error minimization. The latter appeared not to be the case. For the former, I succeeded -if I may say so- exceptionally well. The following blog post [25]:

Is prediction error minimization all there is to the mind

written by Jacob Hohwy, led to the discovery of the Free Energy principle developed by Karl Friston. Meanwhile not only a meeting with the man himself: Karl Friston has taken place but also a full research has been formulated.

I would like to thank my supervisors Ton and Martijn for guiding me and sharing their expertise. Besides all the gained theoretical knowledge, it was extremely valuable to gain some experience in getting all the thoughts inside my head out in the open. I would like to thank Wei Pan for offering me some alternative insights, Ajith for the much-needed check of the derivations and Sherin for the occasional discussion and marks. I am especially grateful for the friends (Mich, An, Fleur, Tess and Mer) that checked parts of my thesis for spelling and clarity. To all those other friends thanks for all the great times I have had during my time studying. It is not common knowledge but worth it to mention. Thanks to all of those who gave me the opportunity to follow a bridging program from a Bachelor Degree in Architecture to a double Master Degree in Mechanical Engineering. I am very happy I was allowed to make this switch. And of course, last but definitely not least, I would like to thank my mama, papa and broertjes for their support.

DELFT, UNIVERSITY OF TECHNOLOGY
NOVEMBER 14, 2018

JULIA ELINA GROENEWEG

CONTENTS

Abstract i

Preface ii

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem Description . . . 3
- 1.3 Contribution . . . 4
- 1.4 Outline . . . 5

2 The Free Energy principle: conceptual review 6
- 2.1 The tendency to veer disorder . . . 6
- 2.2 The Free Energy as upper bound on Surprisal . . . 7
- 2.3 The generative process and the generative model . . . 7
- 2.4 Action selection as an inference process . . . 8
- 2.5 Learning the generative model . . . 9

3 The Free Energy principle: mathematical review 10
- 3.1 The Variational Bayes algorithm . . . 11
- 3.2 The minimization of the Free Energy . . . 13
- 3.3 Discussion and Conclusion . . . 14

4 The Free Energy Principle: State Estimation 17
- 4.1 The relation of FE state estimation with the Kalman Filter algorithm . . . 21
- 4.2 Discussion and Conclusion . . . 23

5 The Free Energy principle: Learning the model matrices 24
- 5.1 The evaluation of FE minimization with respect to the A-matrix . . . 27
- 5.2 The evaluation of FE minimization with respect to the B-matrix . . . 30
- 5.3 Discussion and Conclusion . . . 32

6 A One-Dimensional problem 33
- 6.1 The problem: a Force driven Mass . . . 33
- 6.2 The Model Predictive Control Design . . . 35
- 6.3 The Process of Learning according to the principles of Free Energy . . . 38
- 6.4 The MPC Design embedded with the Free Energy Learning algorithm . . . 45
  - 6.4.1 The simultaneous Learning of the model matrices within the MPC Design . . . 50
- 6.5 Discussion and Conclusion . . . 52

7 A Two-Dimensional Problem 53
- 7.1 The problem: a Pendulum-on-a-Cart . . . 53
- 7.2 The Model Predictive Control Design . . . 55
- 7.3 The Process of Learning according to the principles of Free Energy . . . 58
- 7.4 The Free Energy Learning algorithm . . . 65

7.5 Validation of the Free Energy Learning algorithm by comparison with the SI-technique: Subspace Identification . . . 68
7.6 Discussion and Conclusion . . . 72

8 Discussion **73**

9 Recommendations **75**

10 Conclusion **76**

A MATLAB CODE **77**
A.1 MATLAB Guide . . . 77
A.2 One-Dimensional MATLAB scripts . . . 77
A.3 Two-Dimensional MATLAB scripts . . . 92
A.4 Joint MATLAB scripts. . . . 107
A.4.1 Joint MPC Design scripts. . . . 107
A.4.2 Joint Free Energy scripts . . . 108

Bibliography **113**

List of Figures

3.1 A less generalized form of the Free Energy Principle captured in a Block diagram . . . 16

6.1 The True Process; a Force driven Mass . . . 33
6.2 The generated control input trajectory and estimated output Response by the Model Predictive Control Design implementation . . . 37
6.3 Various realizations of the True Process output Response to the generated control input trajectory compared to the Noise-Free Response . . . 38
6.4 The True Free Energy associated to the True Process . . . 39
6.5 A comparison of the True Process and estimated state according to the principles of Free Energy . . . 41
6.6 A comparison of the True Process Response with the estimated Process Response . . . 42
6.7 A comparison of the True Process and estimated state according to the principles of Free Energy for different realizations . . . 43
6.8 A comparison of the True Process Response with the estimated Process Response for different realizations . . . 43
6.9 A visualization of the Recognition Density and True Posterior . . . 44
6.10 The True Process Response convergence of the control-loop, the True Process Response for VAF-selected iterations . . . 47
6.11 The Free Energy convergence of the control-loop, the Free Energy with respect to model matrix A and B for VAF-selected iterations . . . 48
6.12 The Variance Accounted for percentage convergence for different realizations . . . 49
6.13 The True Process Response of the Model-Control loop generated control input for different realizations . . . 49
6.14 The True Process Response for various realization acquired by the second algorithm which performs control, state estimation and model matrix Learning simultaneously 51

7.1 The true Process; A Pendulum-on-a-Cart problem . . . 53
7.2 The generated control input trajectory and estimated output Response by the Model Predictive Control Design implementation . . . 55
7.3 A comparison of the Performance of Classical and Model Predictive Control using Laguerre functions . . . 56
7.4 The True Process Response where the by MPC design using Laquerre Functions fails to bring the Process Response to the desired Set-Points . . . 57
7.5 A comparison of the by Model Predictive Control generated control trajectory with and without noise compensation . . . 57
7.6 The True Process Response data used in the Free Energy Learning algorithm . . . 58
7.7 The True Free Energy associated to the True Process . . . 59
7.8 A comparison of the True Process and estimated state according to the principles of Free Energy . . . 60
7.9 The convergence of the estimated Process Response simulated with every estimate of the model matrices compared to the True Process Response . . . 62
7.10 A comparison of the True Process Response with the estimated Process Response . . . 63

7.11 A comparison of the True Process Response with the estimated Process Response for different noise levels 64
7.12 A comparison of the True Process Response with the estimated Process Response by the Free Energy Learning algorithm given incorrect initial model matrices 66
7.13 A comparison of the True Process state and state estimation by the Free Energy Learning algorithm with an error in the velocity state variable estimations 67
7.14 The decay of Performance when more deviations are introduced in the initial model matrices provided to the Free Energy Learning algorithm 68
7.15 A comparison of the Subspace Identification estimated Process Response by the MOESP algorithm to the True Process Response and estimated Process Response by the Free Energy Learning algorithm 70
7.16 A comparison of the Subspace Identification estimated Process Response by the PO-MOESP algorithm to the True Process Response and estimated Process Response by the Free Energy Learning algorithm 71

LIST OF TABLES

6.1 The parameters associated with the Force driven Mass problem 33
6.2 The variance of the Process output Response realizations with respect to the Noise-Free Response 38
6.3 The Free Energy value for Sampling Instant: $k = 250$ corresponding to the wrong and correct A matrix 40
6.4 The convergence of the model matrix estimates per Sampling Instant 41
6.5 A quantitative model validation by using the VAF measure for each different realization 44
6.6 Algorithm 1: One-Dimensional MPC Design embedded with the Free Energy Learning algorithm at the end of iteration analogous with Batch Learning 46
6.7 The convergence of the control-loop expressed in numbers 47
6.8 Algorithm 2: One-Dimensional MPC Design embedded with the Free Energy Learning algorithm in an simultaneous manner 50
6.9 The convergence of the control-loop expressed in numbers 51

7.1 The parameters associated with the Pendulum-on-a-Cart problem 54
7.2 The Free Energy value for Sampling Instant: $k = 250$ corresponding to the wrong and correct A matrix 59
7.3 A quantitative model validation by using the VAF measure for each different realization 62
7.4 A quantitative model validation by using the VAF measure for various realizations . . 63
7.5 Algorithm 3: The multi-dimensional Free Energy Learning algorithm in an simultaneous manner with provided control trajectory and True Process Response 65

1
INTRODUCTION

1.1. BACKGROUND

Robotic systems are known to lack adaptability and precision towards uncertain and unknown environments. Humans do own this adaptability and precision while not necessarily equipped with exceptional specifications. The reaction speed of computers for example exceeds that of humans. The brain nevertheless accomplishes tasks in admirable ways [37]. Human intervention improves the robustness of robotic behavior, where robustness is a measure of how good stability is maintained after uncertainty is encountered. The understanding of the human brain -human intelligence- is one of the most appealing challenges facing scientific research. Especially, because it delivers the promise of solving complex problems associated with the increase in complexity of robotic systems and the increase in demand for more autonomous robotic systems [43]. Humans exhibit all properties that are desired for a robotic system in term of adaptive, learning capabilities, compliance, versatility, etc. The next Artificial Intelligence algorithms are believed to become more and more brain-like processes. Over the years many theories of human intelligence have been developed and used for Bio-inspired Control algorithms where a prominent example is discussed in the reference [54]. Currently, a novel theory has gained interest because of its appealing theoretical concepts but most of all because unlike other theories this Free Energy principle proposed by Karl Friston provides a single theory that explains all of the humans behavioral processes. The Free Energy principle makes the issue of behavior computationally tractable by considering all the behavioral processes as various optimization problems with a single objective. The Free Energy principle has a broad reach particularly in the field of Neuroscience. It doesn't therefore surprise Karl Friston to be named most influential neuroscientist back in 2016 [6].

The Free Energy principle will be extensively discussed in the main part of this thesis. For the present, some preliminary remarks on the Free Energy principle will be given to provide for some more background. The key aspect of the Free Energy principle is that it is entirely devoted to predictability. The efficiency that humans possess is according to the Free Energy principle due to excellent predictive capabilities. The single objective of the various optimization problem that can explain human behavioral processes is therefore believed to be minimization of prediction error [16]. The human brain is believed to embody a generative model of the human body and surrounding inhabited environment. This model should be capable of explaining the sensory data retrieved from the sensors of the human body: the eyes and proprioception. If this is not the case, there are two available options. To update the generative model such that the model can explain the sensory data or to actively retrieve new sensory data that the existing model can explain. From an Engineering standpoint, this bluntly translates into the option of the process of System Identification or the option of the process of control. The process of System Identification represents the art and science of designing mathematical models of dynamical Processes from observed input and output signals [30]. The process of control represents the generation of a control input trajectory that brings the output signal of a dynamical Process to any desired reference [38].

The Free Energy principle suggests a control algorithm that performs dualistic prediction error minimization where the robotic system has a choice to eliminate prediction error either by an update of their model or by applying a suppressing controller. Such a control algorithm does not yet exist while it offers the potential to improve the discussed adaptability and precision issues of robotic systems facing uncertain and unknown environments.

Improvement of the predictive model is not an unfamiliar topic in the field of Control Engineering. The purpose of any model is to capture salient trends or regularities in observed data with a view to the prediction of future events. These predictions of future events are used to generate control input trajectories [39]. The more accurate the predictions are -the model is well specified- the more controllable robotic systems become. For a majority of robotic applications however either the observed data is far too complex or the underlying process is not well enough understood [4]. Often, the model is only a simplifying approximation of the true Process designed with physical insight and expertise of the engineer. These approximations are accompanied by the risk of being too simplistic and only to perform well in severe controlled environments. In other words, the models are too simplistic to deal with any uncertainty present in daily life. Engineering dynamics, nowadays, have largely assumed deterministic models to be appropriate for system modeling and prediction however events have suggested otherwise [55]. Especially, robotic systems immersed in highly dynamical environments demand for more adaptive and online System Identification approaches [38]. Novel approaches to System Identification are found in the field of statistics or more specific in the field of Machine Learning. The field of Machine Learning offers alternative purely data-driven approaches based on probability theory. The Reinforcement Learning algorithm is an example of a Machine Learning approach more widely known among control engineers [30]. A disadvantage of Reinforcement Learning algorithm is that they are not always straightforward to reproduce [39]. The Free Energy principle represents an adaptation of the Machine Learning approach: Variational **E**xpectation **M**aximization (or Variational Bayes). A less well-known approach among control engineers but a more intuitive -providing more physical insight- algorithm in respect to Reinforcement Learning. In this thesis, therefore concepts from three different fields -the Neuroscience, Machine Learning and Control Engineering field- will be discussed. From this part, it can be concluded that the choice of “eliminating prediction error by an update of the robotic systems model” is precisely what the process of System Identification represents. The novelty of the Free Energy principle lies in the second choice of “eliminating prediction error by applying a suppressing controller”.

1.2. PROBLEM DESCRIPTION

The previous section served to explain briefly why the Free Energy principle is promising enough for further research. The general aim of the research group I am currently part of is to transform the biological and mathematical concepts of the Free Energy principle into an implementable Bio-inspired Control algorithm that can be used to control complex robotic applications. The objective is to improve the overall but specifically predictive performance of robotic control. Time will tell whether the implementation of the principles of Free Energy will indeed yield favorable results. Currently, the primary problem is the theory to be complicated. The degree of difficulty of the available material on the Free Energy principle is high. The reference [32] sets an example that backs up this statement. The principles of Free Energy are therefore not straightforwardly implementable. Problems encountered are for example the theory to be presented over different stages of evolution and the use of various notations [8]. Furthermore, the theory is presented with a biological implementation in mind which is inherently different from a robotic implementation. While there are parallels, the theory lacks several Control Theory essentials.

The purpose of this thesis is to provide an increase in understanding of the Free Energy principle. The following reference [8] has a similar objective and can be regarded as a valuable contribution. This reference does not address the process of Learning a dynamical model. The dynamical model of Process is assumed given. The Learning of the dynamical model is in the material on the Free Energy principle also often a neglected topic. This is however not surprising. The dynamical model of humans is believed to mostly be a product of natural selection. This thesis will specifically address the process of Learning dynamical models according to the principles of Free Energy. Rather then a biological implementation a robotic implementation is considered. This requires an evaluation of the principles of Free Energy with respect to a Control Theory perspective.

The primary research question of this thesis is: **How does the Learning of dynamical models according to the principles of Free Energy relate to concepts from Control Theory?**

The research question is assessed from various angles. The first sub-question is: **1. What information is known about the Learning of dynamical models, what are the primary considerations and assumptions and are these familiar in Control Theory?** The short answer is the Free Energy principle to be an adaptation of the Machine Learning methodology known as Variational EM. The EM algorithm employs wide popularity in a broad variety of fields of applied statistics. Within Control Theory the EM algorithm is however not well understood and utilized [24]. The EM algorithm is often presented in an abstract setting that can be applied quite generally but makes it less attractive [36]. The methodology entails the iterative performance of an Expectation Step -state estimation- and a Maximization Step -Learning of the model- until some kind of information measure is optimized. The Free Energy principle is entirely devoted to gain a full set of information: the state variables, the input trajectory and model parameters. Once this set is obtained, the making of exact predictions of the Process Response becomes easy. The second sub-question is: **2. What entails state estimation according to the principle of Free Energy?** where the short answer is the Kalman Filter. The third sub-question is: **3. What entails model Learning according to the principles of Free Energy?** where the short answer is Least-Square Regression. The derived equations together form the Free Energy Learning algorithm. The final sub-question is: **4. How can this Free Energy Learning algorithm be implemented on typical control problems and what are the conclusions to draw from the performance of these implementations?** Where the short answer is the Free Energy Learning algorithm to yield accurate predictions. In case of all the short answers, there are many considerations involved discussed in detail throughout the thesis.

The Free Energy Learning algorithm requires a control signal which ultimately should be a control signal generated according to the principles of Free Energy. Because it is not yet entirely specified how the control part of the Free Energy should be correctly implemented the derived Learning algorithm will be implemented in combination with a Model Predictive **C**ontroller. A MPC Design employs models, traditionally obtained from Mechanical Engineering insights, to predict the effect of future reactions of the Process output on the applied control signal. The control signal is obtained by minimization of a cost function. The performance of the Control Design depends on how well the dynamics of the system are captured by the model. The incorporation of Learning the dynamical models could considerably improve the MPC Design [43]. MPC Design and the Free Energy principle have, in this sense, the similar objective of obtaining a as good as possible predictive model.

The Process and model -with prescribed input signals like Force or a Voltage and output signals like position or velocity- will be described by a state space representation. Similar, in the papers written by K. Friston the Generative Process and Generative model are also described by a state space representation, although in a more generalized form. State space representations have shown to be a powerful tool to analyze and control the dynamics of a Process. Furthermore, are computations more numerically reliable when done based on a state space representation [49]. A state is a set of state variable like the position or velocity which are all the variable -all the information- required to predict the future. A state is not a decision variable such as model parameters but is completely determined by the previous state, control input and parameters. The parameters are values like the length or Mass which are often constant [39].

1.3. CONTRIBUTION

The contributions in the research field of this thesis are summarized as:

- A piece of understanding in the Free Energy principle evaluated from a Control Theory perspective. Specifically, a piece of understanding in the Learning of dynamical models according to principles of Free Energy.
- A set of the prime considerations and assumptions associated with the Free Energy principle in relation to their effect on the implementation of the Free Energy Learning algorithm.
- A proof the minimization of the construct Free Energy with respect to the state, under the assumptions specified, yields to equations equivalent to the equations of the Kalman Filter algorithm.
- Improving the acceptance and utilization of the Machine Learning methodology: Variational EM in Control Theory by the implementation of the Free Energy Learning algorithm on typical Control problems.

1.4. OUTLINE

The main matter of this thesis is divided into six's chapters that discuss the Free Energy principle in the following ways:

- **Chapter 2:** provides the necessary background information on the concepts of the Free Energy principle. This chapter does not yet include any mathematical notations and purely serves as a conceptual review.
- **Chapter 3, 4 and 5:** provides -on the contrary to the previous chapter - a mathematical review of the Free Energy principle. All of the equations part of the Free Energy Learning algorithm are derived within these chapters. In chapter 3, the mathematics of the Free Energy in general will be discussed. In chapter 4, the mathematics of the process of state estimation according to the principles of Free Energy will be discussed. The relation of the Free Energy state estimation with the well-known Kalman Filter algorithm will require a substantial part of this discussion. In chapter 5, the mathematics of the process of learning the model according to the principles of Free Energy will be discussed.
- **Chapter 6:** includes the first implementation of the derived algorithm on a simple one-dimensional control problem. First, the Free Energy Learning algorithm is tested given a quite extensive set of correct data. This corresponds to a case where the Free Energy Learning algorithm is assumed to converge. The second part of the chapter challenges this assumption of convergence. The Free Energy Learning algorithm is given random initial data. In this chapter, the MPC Design is incorporated into the Free Energy Learning algorithm such that a closed control-loop results.
- **Chapter 7:** includes the second implementation of the derived algorithm on a complex - highly unstable- control problem: a Pendulum-on-a-Cart. Because of the complexity of the problem, the MPC Design implementation is slightly more advanced. The basic MPC Design of chapter 6 is expanded with Laguerre functions. A similar approach as in chapter 6 is followed. First, the Free Energy Learning algorithm is tested assuming convergence and in the second part, this assumption of convergence is once again challenged. In this chapter, in contrast to chapter 6 the MPC design is not incorporated into the Free Energy Learning algorithm. The Free Energy Learning algorithm is given a batch of input and output data. The same batch is given to a similar System Identification approach: the Subspace Identification approach. A comparison of the performance of both algorithms is made.

The last chapters of this thesis include the discussion, recommendations and conclusion. In the Appendix, all developed MATLAB scripts can be found.

2

THE FREE ENERGY PRINCIPLE: CONCEPTUAL REVIEW

This chapter serves as a summary of the key aspects of the by K. Friston proposed and developed Free Energy principle; an unifying brain theory.

The Free Energy principle is an attractive theory that can explain different brain functions with a single framework. The Free Energy principle delivers an unifying scientific explanation of all behavioral aspects in contrast to current brain theories that only explains specific parts [20]. In the Free Energy principle, the brain is viewed as an Inference machine that actively explains and predicts the sensations that humans encounter. The mathematical equivalent of sensations is the dataset y. This dataset stores the sensory data that is received by sensors. In case of humans, the sensory data comprises the data received by the sensors of the human body: the eyes and proprioception [13]. The following section justifies why it is important for the brain to be capable of explanation and prediction of sensory data.

2.1. THE TENDENCY TO VEER DISORDER

The central idea behind the theory derives from the field of Thermodynamics from which is known that a tendency towards veer disorder exists. The Free Energy principle is a mathematical formulation of how adaptive agents -such as humans- resists this natural tendency to disorder [13]. Adaptive agents restrict themselves to a limited number of states. The state encloses the 'amount of information' that, together with the input, uniquely determines the behavior of a Process. Surprising states are the states that are unenclosed in the probabilistic model that adaptive agents maintain of their typical, habitable environment [8]. As a result of stochastic and chaotic perturbations, the adaptive agent will experience states that are atypical and therefore surprising. A perturbation is, for example, any external forces affecting the adaptive agent such that a state transition is enforced. An attractor state is a state prescribed and enclosed by the probabilistic model and therefore is the least surprising. Most environments however do not support stable fixed-point attractor states [15]. A necessary condition for a biological agent to exist is to adopt a policy that minimizes Surprisal [22]. By minimization of Surprisal, the adaptive agent ensures the states to evolve according to an expected flow. In the following section, the relation between the minimization of Surprisal and the minimization of Free Energy will be discussed. The notion of Free Energy, an information-theoretic construct, derives from Statistical Physics and is widely used in the field of Machine Learning [20]. The Free Energy principle combines advanced concepts from several fields including Statistical Physics, Probability theory, Machine Learning and theoretical Neuroscience [8].

2.2. THE FREE ENERGY AS UPPER BOUND ON SURPRISAL

Surprisal is a measure of how good a generative model of a Process can explain the sensory data received from the sensors of that same Process. The minimization of Surprisal is therefore identical to the maximization of the sensor evidence that supports the generative model. The generative model depends from agent to agent, what is surprising for one agent may not be surprising for another [13]. The mathematical equivalent of surprise is the negative log-Likelihood of the observed sensory data: $-\ln p(y_k)$ [22]. The measure of surprise cannot be quantified by adaptive agents because it entails a marginalization over the yet unknown causes of sensory data [15]. It is possible to minimize a bound on Surprisal. The Free Energy represents this bound. Minimization of the Free Energy can either be accomplished by an update of the generative model or by active manipulation of the sensory data. To perform these processes the adaptive agent has to gain some notion of the causes of sensory data. This process of deducing properties from underlying probability distribution through analysis of data is referred to as inference [48]. The causes of sensory data can be regarded as a set of variables. The causes set is partitioned into the state, model parameters, hyper parameters and the input. The values of this set of variables are updated such that the Free Energy is minimized and at the same the time the model evidence is maximized. To summarize adaptive agents are considered to be little Free Energy machines that garner and model sensory information to maximize the evidence of their own existence [13].

2.3. THE GENERATIVE PROCESS AND THE GENERATIVE MODEL

From the brains point of view, the environment includes both the internal and external milieu, the body and its surroundings. All the processes that are taking place outside the brain constitute the environment [8]. The dynamic behavior of the environment is described by equations of motion which can specify any trajectory part of the entire state space according to Physical Laws [13]. A generative process represents a specific fraction of all the possible equations of motion. Adaptive agents are examples of generative processes. Sensory data is acquired from the generative process. By selection of specific input sequences applied to the generative process, the sensory data can be actively manipulated [8]. Adaptive agents can, for example, change their configuration and as a consequence enforce a state transition. The state is part of the set of variables that represent the causes of sensory data. The selected inputs depend on beliefs about what the next state or even state trajectory will be. Beliefs are incorporated in the generative model [18]. From the previous section, it is known that the brain embodies a generative model for which it tries to gather the greatest evidence. The generative model represents a probabilistic, forward model that maps expected causes to their observed consequences [16]. The generative model describes what adaptive agent beliefs the generative process to be [21]. The specific input sequences are only applied to the generative Process and are not part of the generative model. Instead, the generative model incorporates specific control policies that determine the desired behavior and represents the beliefs. The equations that actually generate the data -the generative process- and those employed by the generative model are not necessarily the same. Adaptive agents can have a different belief about what the generative process represent relative to what the generative process actually represents. The selected input tries to conceal this discrepancy [15]. This means that the process of control -specification of the input sequences- is an inference problem rather than a conventional optimal control problem [19]. This will be discussed in the following section.

The equations assumed to actually generate the sensory data, the generative Process, are:

$$x_{k+1} = \mathbf{f}_{\text{True}}(x_k, u_k, \theta) + w_k \tag{2.1}$$
$$y_k = \mathbf{g}_{\text{True}}(x_k, \theta) + z_k \tag{2.2}$$

The equations that generate the estimated data, the generative model, are:

$$(\mu_x)_{k+1} = f_{\text{Model}}((\mu_x)_k, \zeta, \langle\theta\rangle) \tag{2.3}$$
$$\hat{y}_k = g_{\text{Model}}((\mu_x)_k, \langle\theta\rangle) \tag{2.4}$$

Where x represents the state sequence(s), μ_x represents the expectation of the state sequence(s), θ represent the Process parameters, $\mathbb{E}[\theta]$ represents the model parameters, $\hat{y}$ represents the prediction of the output sequence(s), u represents the input sequence(s), ζ represents the prior expectation or the beliefs, w represents the Process Noise sequence(s) and z represents the Measurement Noise sequence(s).

2.4. ACTION SELECTION AS AN INFERENCE PROCESS

In a classical optimal control problem, the process of control -action selection- and the process of Filtering in combination with System Identification -belief optimization- are usually performed as two individual processes. Conversely, in the Free Energy principle action selection calls for the same inference scheme used to optimize beliefs [21]. The external cost functions in optimal control are replaced by prior beliefs about state transitions [22]. The goals and preferences are framed in terms of prior beliefs incorporated into the generative model [21]. Both approaches should yield to similar or even equivalent results. The duality between optimal control and estimation has already been acknowledged by engineers such as T. Todorov and R. Kalman. The minimization of a cost function can been seen as equivalent to the maximization of the Likelihood of desired observations [14]. An optimal policy in terms of a probability distribution on desired states offers a more simple and flexible approach of the control problem [15]. A classical control problem requires a forward and inverse model. The calculation of the inverse model that maps the desired trajectories to controlled state changes is often hard. Conversely, an inference control problem only requires a forward model: the generative model.

Both the process of action selection referred to as active inference and the process of prediction updating referred to as perceptual inference are driven to facilitate the restriction of adaptive agents to a limited number of preferred outcomes [21]. Where the process of perceptual inference is passive, the process of active inference ensures resistance to dispersion of states by random fluctuations [22]. The Process Response of generative processes prescribed by the Free Energy principle are promised to be remarkably robust to noise and exogenous perturbations. These uncertainties are not necessarily perceived but explained away by actions. The generative model is not updated to incorporate all uncertainty that is encountered by adaptive agents [15]. Rather the generative model is updated specifically in controlled environments. In the following section the obtaining of the generative model in a controlled environment will be discussed. The action is selected so that the hard constraints encoded in the generative model are satisfied [8]. The selected action therefore realizes prediction biases towards preferred outcomes [19].

2.5. LEARNING THE GENERATIVE MODEL

The previous sections served as a conceptual background to the main focus of this thesis: the process of learning the generative model. From the previous sections is it known the generative model to incorporate an optimal policy. This optimal policy -the beliefs- are induced by the learning of optimal trajectories in a controlled environment [15]. The adaptive agent is immersed in a controlled environment that furnishes an optimal policy. For adaptive agents -like humans- the process of learning is primarily a process of evolution and natural selection. For robotic applications the process of Learning does not have to depend on these -long time scale- concepts. In the following chapters the process of learning will be extensively discussed. To summarize the adaptive agents learns or updates any causal structure of a controlled environment. The learned equations of motion -the generative model- describes divergence free motion [14]. When the agent is placed back into an uncontrolled environment the beliefs incorporated in the generative model are retained by input selection.

3

THE FREE ENERGY PRINCIPLE: MATHEMATICAL REVIEW

"*Pierre Laplace: Probability theory is nothing but common sense reduced to calculation.*"

In this chapter, the mathematics behind the Free Energy principle will be discussed in general. The chapter primarily serves to substantiate why it is allowed to minimize Free Energy and how this yields to clear-cut optimization problems. The chapters 4 and 5 will specifically discuss the mathematics associated with the process of Learning the generative model. At the end of this chapter, it will be made clear why this requires two separate chapters.

Let's start with the mathematical equivalent of Surprisal, the negative log-Likelihood of the observed sensory data $-\ln p(y)$. The Likelihood distribution expresses how probable the observed sensory data is for different settings of the causes. The Likelihood is a typical term used in Probability theory. Probability theory provides a consistent framework for the quantification and manipulation of uncertainty [5]. Uncertainty and Surprisal bear a similar meaning. Remember that a necessary condition for the existence of any adaptive agent is to adopt a policy against Surprisal. A Bayesian interpretation of Probability theory provides the means required to minimize Surprisal. Bayesian theory allows for a quantification of Surprisal, for precise revisions of Surprisal in light of new sensory data and for taking optimal actions [5]. These operations correspond to the two inference processes that where presented in chapter 2 capable to accomplish minimization of Surprisal. Perceptual inference represents the process of model updating such that the model is in better correspondence with the observed sensory data. Active inference represents active manipulation of the sensory data such that the observed sensory data is in better correspondence with the model. Both operations maximize the Likelihood -the model can explain all sensory data- and therefore by the definition lead to a minimization of Surprisal: $\max p(y) = \min -\ln(p(y) = \min \ln(1(y))$. Before proceeding to the mathematical definition of the Likelihood a brief note on the use of the log-Likelihood rather than the Likelihood.

The reason to use the log-Likelihood is because this will significantly simplify the upcoming calculations. The use of the log-Likelihood is allowed because the problems considered are all optimizations. A logarithmic function is monotonically increasing. The maximum of the Likelihood therefore is equivalent to the maximum of the log-Likelihood. Similar, the negative logarithmic function is monotonically decreasing. The maximization of the Likelihood therefore is equivalent to minimization of the negative log-Likelihood [29].

Minimization of Surprisal is accomplished once a Maximum Likelihood solution is found for the causes: ϑ^*. The causes set is partitioned into the state, model parameters, hyper parameters and the input: $\vartheta = \{x, \theta, \lambda, u\}$ [27]. The Likelihood is a function of the causes. In Continuous-Time the Likelihood is defined as [5]:

$$p(y) = \int d\vartheta p(y, \vartheta) = \int d\vartheta p(\vartheta) p(y|\vartheta) \tag{3.1}$$

Where $p(y, \vartheta)$ is known as the joint probability. The product rule is used to describe the relationship between the joint and conditional probability: $p(x, y) = p(y|x)p(x) = p(x|y)p(y)$ [5].

To obtain an exact maximum Likelihood solution one has to integrate over all possible settings of all known and unknown causes. This is challenging, if not impossible. The integral part of the Likelihood presented in equation 3.1 is often intractable [4]. Variational methods consider a close-to-exact solution and avoid calculation of the exact solution. The variational method used to minimize Surprisal is known as the Variational Bayes algorithm. The Variational Bayes algorithm represents a Variational implementation of the Expectation Maximization methodology.

3.1. THE VARIATIONAL BAYES ALGORITHM

The Variational Bayes algorithm is used to find the Maximum Likelihood solution. The Maximum Likelihood solution best explains the observed sensory data y. Often, it is convenient to consider a representation of initial knowledge. Initial knowledge represents the already available knowledge of the problem. One of the strengths of the Variational Bayes algorithm is that the inclusion of prior knowledge arises naturally. Prior knowledge is implemented by the use of Prior distributions with almost no alterations required in the algorithm [17]. The Prior distribution represents the probability available before any observations [5]. If Prior distributions are included instead of the Maximum Likelihood problem represented by equation 3.2, the Maximum A Posteriori problem represented by 3.3 is considered [4].

$$\vartheta_{ML} = \max_{\vartheta} p(y|\vartheta) \tag{3.2}$$

$$\vartheta_{MAP} = \max_{\vartheta} \frac{p(y|\vartheta)p(\vartheta)}{p(y)} \tag{3.3}$$

The Posterior distribution captures all information inferred from data about the causes and is defined as $p(\vartheta|y)$. The Bayesian treatment to compute the Posterior distribution given observed sensory data is to first select a Prior distribution and subsequently use Bayes theorem [5]. According to Bayes rule, the Posterior is defined as:

$$\underbrace{p(\vartheta|y)}_{\text{Posterior}} = \underbrace{p(y|\vartheta)}_{\text{Likelihood}} \times \underbrace{p(\vartheta)}_{\text{Prior}} \tag{3.4}$$

The key aspect of variational methods is the approximation of difficult mathematics with more simpler and tractable forms of mathematics [34]. The Variational Bayes algorithm approximates the Posterior distribution by imposing an optimized bound on the Posterior employing an iterative and similar scheme to the **E**xpectation **M**aximization algorithm [4]. The observed sensory data represents a function of all variables contained in the causes set. For the state there exist an additional mapping: $y(x) \rightarrow x$. The state together with the input represents the complete data set required to compute the model (the Posterior or Likelihood distribution). The observed sensory data is equal to the state in the case that the data is fully observed. In most cases, sensory data is only partially observed [9]. The EM algorithm exploits that if data were fully observed it would be easy to compute the model parameters and therefore easy to compute either a ML or MAP solution [34].

The EM algorithm consists of two consecutive steps where the algorithm in an iterative manner alternates between. The E-step forms a conditional expectation over the data that is missing -the state- given observed data and the current estimate of the model parameters. The M-step maximizes the conditional expectation with respect to the model parameters [9].

The approximation of the true Posterior is defined as $q(\vartheta)$ referred to as the recognition density [8]. A useful measure is the Kullback-Leibler divergence which is a measure of how one probability distribution is different from a second reference probability distribution. Henceforth, the mathematics will be in Discrete-Time formulation which mainly implies a shift from integrals to summation [41]. Later in the chapter this will be more convenient. The Kullback-Leibler divergence between the Posterior $p(\vartheta|y)$ and the recognition density $q(\vartheta)$ is defined as [5]:

$$D_{KL}(q(\vartheta)||p(\vartheta|y)) \equiv \sum q(\vartheta) \ln\left(\frac{q(\vartheta)}{p(\vartheta|y)}\right) \tag{3.5}$$

Using the product rule and the logarithm laws $\ln(x/y) = \ln(x) - \ln(y)$ and $\ln(xy) = \ln(x) + \ln(y)$, the Kullback-Leibler divergence is extended to the following -very useful- equation 3.6:

$$\begin{aligned} D_{KL}(\cdot) &= \sum q(\vartheta) \ln\left(\frac{q(\vartheta)}{\frac{p(y,\vartheta)}{p(y)}}\right) \\ &= \sum q(\vartheta) \ln(q(\vartheta)) - \sum q(\vartheta) \ln(p(y,\vartheta)) + \sum q(\vartheta) \ln(p(y)) \end{aligned} \tag{3.6}$$

The Kullback-Leibler divergence is a function of the expected recognition density, the expected joint probability distribution and the expected Likelihood distribution [27]. Expectation represents the average value of some function $f(x)$ under a probability distribution $p(x)$ and is given as $\mathbb{E}[f] = \sum p(x) f(x)$ [5]. The Kullback-Leibler divergence can be written as a function of the negative log-Likelihood -Surprisal- and remaining terms which are referred to as Free Energy. This is the same Free Energy construct like the one where K. Friston refers to in his Free Energy principle. Ordinarily, the Free Energy is not explicitly named in relation to the EM algorithm or its variational equivalent the Variational Bayes algorithm. Implicitly the Free Energy is always part of the EM algorithm. In the regular EM algorithm, the conditional expectation formed and maximized is the Likelihood (or Posterior) distribution. In the variational EM algorithm, the conditional expectation formed and maximized is the Free Energy [4] [35].

$$D_{KL}(\cdot) = \underbrace{\mathbb{E}[\ln(q(\vartheta)] - \mathbb{E}[\ln(p(y,\vartheta)]}_{\text{Free Energy}} - \underbrace{\mathbb{E}[-\ln(p(y)]}_{\text{Surprisal}} \tag{3.7}$$

It all comes together here: the Free Energy serves as a bound on the negative log-Likelihood because of Gibb's inequality which states that the KL-divergence always is positive: $D_{KL}(Q||P) > 0$. The minimization of Free Energy therefore yields minimization of Surprisal [3]. If the KL-divergence is equal to zero; the Free Energy becomes the log-Likelihood. The question that arises is whether the Free Energy principle is just an implementation of the Varational Bayes (variational expectation maximization) algorithm which it is not although there are quite some parallels. The use of the ϑ suggests a different take on the EM algorithm. This -amongst others- will be discussed in the following section.

3.2. THE MINIMIZATION OF THE FREE ENERGY

The most general form of the Free Energy equation is given as:

$$\mathscr{F} = -\mathbb{E}[\ln p(y,\vartheta] + \mathbb{E}[\ln q(\vartheta)] \tag{3.8}$$

This equation 3.8 serves as the objective function of the specified optimization problem to be minimized. For the non-variational case, the Free Energy is the log-Likelihood, there will be minimized with respect to ϑ directly. For the variational case, there will be minimized with respect to $q(\vartheta)$.

The causes set ϑ contains multiple quantities which most likely have a different temporal scale. The Mean-Field assumption allows for the factorization of the recognition density $q(\vartheta)$ [3]. The dependencies between the different quantities can be ignored which significantly simplifies the minimization of the Free Energy [12]. The Free Energy can be separately minimized with respect to each of the factors [5].

$$q(\vartheta) = \prod_i q(\vartheta^i) \tag{3.9}$$

The Mean-Field assumption leads to the natural partition of the causes into the state, the model parameters, the hyper parameters and the input: $q(\vartheta) = q(x)q(\theta)q(\lambda)q(u)$ [17]. The four separate evaluations of the Free Energy equation 3.8 are:

$$\min_{q(x)} \mathscr{F}(x) \qquad \min_{q(\theta)} \mathscr{F}(\theta) \qquad \min_{q(\lambda)} \mathscr{F}(\lambda) \qquad \min_{q(u)} \mathscr{F}(u) \tag{3.10}$$

Typically, only the first two problems of equation 3.10 are considered in the Variational Bayes algorithm. The reference [4] also considers the third problem of equation 3.10. Conversely, the last problem of equation 3.10 is hardly ever considered. The input is typically either arbitrarily selected or in case of control applications found by a control design. This represents the in chapter 2 presented difference between the conventional optimal control problem and the inference control problem. Every problem of equation 3.10 depends on the result of the other problems. An iterative scheme therefore is required. In relation to the Expectation and Maximization distinction, one would consider the state and input minimization problems to be part of the expectation step. The state and input are the missing variables required to find the model variables: the model and hyper parameters. Once the state and input are known the estimation of the model and hyper parameters becomes easy [41].

The most simple solution of finding each individual minimum is to set the derivative to zero.

$$\delta_{q(\vartheta^i)} \mathscr{F} = 0$$

The minimization problems presented in equation 3.10 transform into the following problems:

$$\frac{\delta \mathscr{F}(x)}{\delta q(x)} = 0 \qquad \frac{\delta \mathscr{F}(\theta)}{\delta q(\theta)} = 0 \qquad \frac{\delta \mathscr{F}(\lambda)}{\delta q(\lambda)} = 0 \qquad \frac{\delta \mathscr{F}(u)}{\delta q(u)} = 0 \tag{3.11}$$

In the introductory chapter 1 is specified the focus of this thesis to be on the Learning of dynamical models. The input sequence is from now on assumed given by a conventional optimal control design. The hyper parameter will be briefly mentioned in chapter 5 but in further implementations neglected. The remaining problems presented in equation 3.11 will be the topic of the upcoming chapters.

$$\frac{\delta \mathscr{F}_x}{\delta q(x)} = 0 \qquad \frac{\delta \mathscr{F}_\theta}{\delta q(\theta)} = 0 \tag{3.12}$$

3.3. DISCUSSION AND CONCLUSION

It has to be noted that according to K. Friston the problems laid out in equation 3.11 are assumed to be **not** tractable [17]. The reason for this is not explicitly mentioned. One can assume it is related to the development of the Free Energy principle with a biological implementation in mind. Adaptive agents could be considered as a Process situated in a highly stochastic environment. To cope with these stochastic dynamics, K. Friston introduces the Fokker-Planck equations to describe the dynamical behavior of a Process [17]. The Fokker-Planck equation is a Partial Differential equation oppose to the Ordinary Differential equation normally used. A Partial Differential Equation describes the time evolution of the probability density function of the velocity of a particle under the influence of drag and/or random forces [44]. The derivation of a Partial Differential Equation provides information on the probability density function of the stochastic process. Therefore, rather than one static solution an ensemble of solutions is evaluated. The concept of Partial Differential Equations is rather complex.

Another concept introduced by K. Friston is the incorporation of generalized motion. The concept of generalized motion supports a conditional density over paths -state trajectories- opposed to only over a single state [17]. This is roughly translated in that usually only the first and second order time derivative -the position and velocity- of a Process are specified. The use of generalized motion means that higher order time derivatives are taken into account as well $\mathbf{x} = [x, x', x'', x''', \ldots, x^p]$. In reality, these higher derivatives referred to as generalized motion always exist. In the standard modeling approaches generalized motion is however almost always ignored. Ignoring is a lot more convenient and still furnishes reasonable approximations. The standard state space representation actually is a special case of the generalized state space representation where the higher order motion of random fluctuations is ignored. Predominantly, the noise affecting the Process is assumed to be a Wiener process. In Wiener processes, the noise derivatives have no precision and are therefore neglected. It is believed that the assumption of the noise being a Wiener process is less plausible for biological or other systems where random fluctuation is themselves generated by dynamical systems. The assumption of the noise being a Wiener process is believed to not be appropriate to resemble real fluctuations [12].

The concepts both are in general unknown to most engineers with as main reason that the concepts are complex and so far are not deemed necessary. In the introductory chapter 2 the desire for robotic systems to gain better predictive capabilities or in other words to gain a better predictive model is mentioned. The Machine Learning part of the Free Energy principle, the use of Variational EM should already gain a significant improvement. Variational EM has made a name in the statistics field it is however relatively unfamiliar in the field of Control Engineering relative to the System Identification techniques **P**rediction **E**rror **M**inimization and Subspace Identification. From the concepts of Partial Differential Equations and generalized motion, it is unknown whether a significant improvement will be achieved however one can guess a better representation of the fluctuations will probably result in a better predictive model. Because of the complexity of the concepts, the concepts will not be implemented but will be referred to in the remaining chapters of this thesis. The assumed non-tractability of the problems presented in equation 3.12 is debatable. The implementations are not bound to biological implementations but rather to robotic implementations. For a first implementation, a simple optimization will suffice. In the case of more complex implementation, it is helpful to keep different optimization methods -like gradient descent- in mind.

In summary the main difference between the Free Energy principle and Variational EM are the incorporation of the input -the input trajectory is found by a similar optimization problem as the state or model parameters (equation 3.10)- and the incorporation of generalized motion. Because the incorporation of generalized motion will not be considered yet and a simple optimization method will assumed to be tractable the evaluation of the optimization problem presented in equation 3.12 will be conform to the Variational EM algorithm. The chapters 4 and 5 addressing the evaluation of the optimization problem presented in equation 3.12 and the chapters 6 and 7 addressing the implementations will therefore be conform to the Variational EM algorithm as well.

In figure 3.1 the Free Energy principle is captured in a block diagram. This block diagram assumes there to be a solution to the problems presented in equation 3.11. The optimization problems are assumed to be tractable. The block diagram therefore represents a less generalized version of the Free Energy principle. The grey area represents the process of Learning. The block diagram assumes the process of Learning to be done on a controlled signal generated by a **M**odel **P**redictive **C**ontrol Design. In the future, this MPC Design will presumably be replaced by a process similar to the processes present in the upper half of the block diagram. All of the variables between the $\langle\rangle$ brackets represent an expectation. The variable ζ represents a prior expectation which could be seen as a desired control policy. How this prior expectation is realized and what precisely the connection is with the controlled environment -the gray area- is not concluded yet into an explicit understanding. The variable r_k is a selected reference set point on which the application is trained on. In the chapters 6 and 7 the MPC Design will be further specified. The block diagram provides an overview of the processes involved in the Free Energy principle. Furthermore, the block diagram presented in figure 3.1 shows the role of the process of Learning the dynamical model relative to the process of control. The Generative process and process of state estimation are part of both the process of Learning the dynamical model and the process of control. The switch symbol represents which process is currently active.

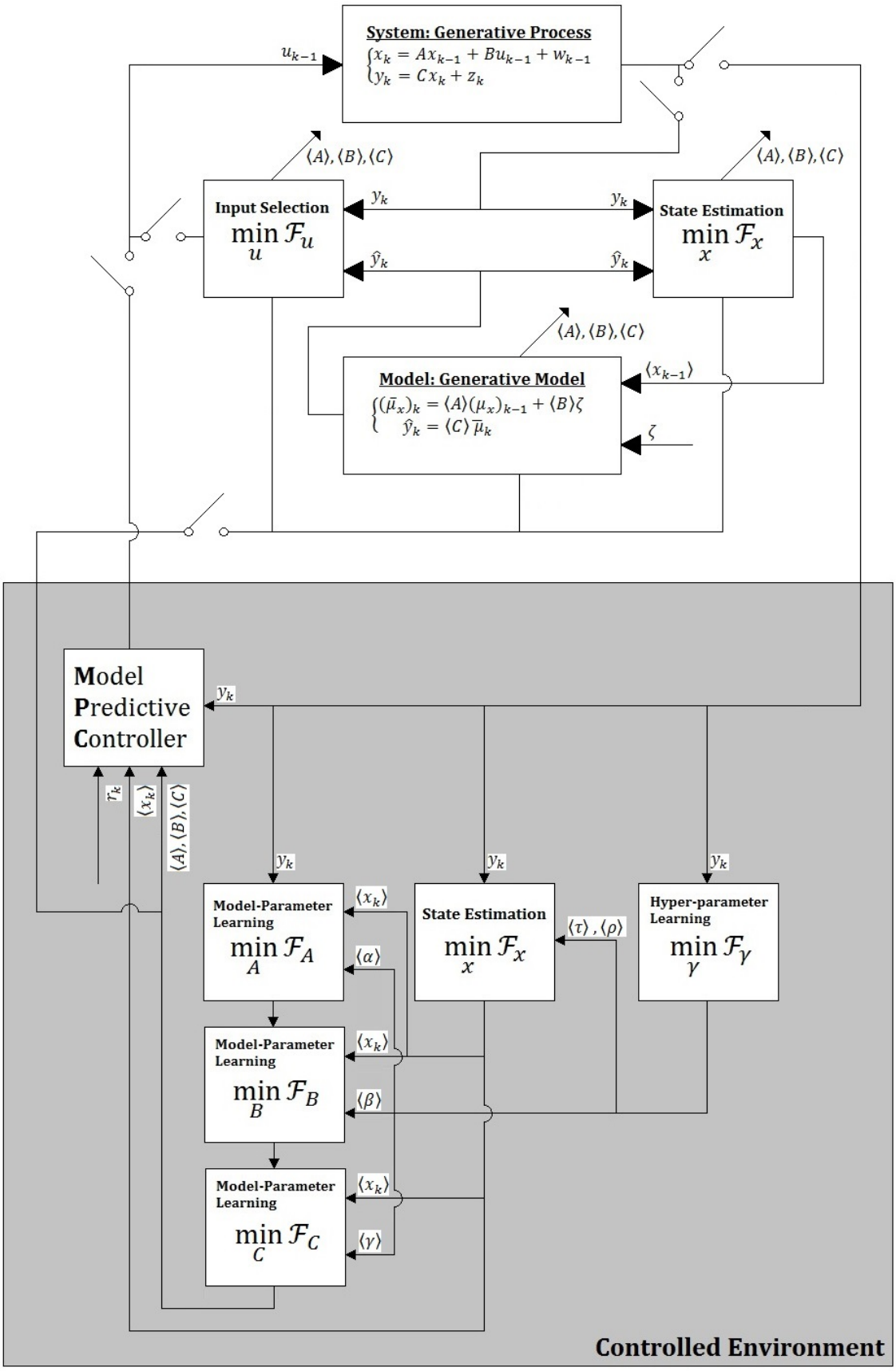


Figure 3.1: A less generalized form of the Free Energy Principle captured in a Block diagram

4

THE FREE ENERGY PRINCIPLE: STATE ESTIMATION

As discussed in the previous chapter 3 the Free Energy principle is an adaptation on the Variational EM algorithm. In the variational EM algorithm, an Expectation and Maximization step are iteratively performed until an information measure is optimized. The Free Energy equation represents this information measure. This chapter will evaluate the first problem of equation 3.12 confirm to the Expectation step of the variational EM algorithm.

Remember, the Free Energy equation to be specified as [17]:

$$\mathscr{F} = -\mathbb{E}[\ln p(y,\vartheta)]_{q(\vartheta)} + \mathbb{E}[\ln q(\vartheta)]_{q(\vartheta)} \tag{4.1}$$

The causes set ϑ is partitioned into the state x, the model parameters θ, the hyper parameters λ and the input u. By substitution of the causes set the Free Energy equation 4.1 extends to:

$$\mathscr{F} = -\mathbb{E}[\ln p(y,x,\theta,\lambda,u)]_{q(x,\theta,\lambda,u)} + \mathbb{E}[\ln q(x,\theta,\lambda,u)]_{q(x,\theta,\lambda,u)} \tag{4.2}$$

The Mean-Field assumption allows for a separate evaluation of each of the partitions. In this chapter, the emphasis is on the evaluation of the state partition. The Free Energy equation with respect to the state x is defined as:

$$\mathscr{F}_x = -\mathbb{E}[\ln p(y,x)]_{q(x)} + \mathbb{E}[\ln q(x)]_{q(x)} = \prod_{k=0}^{K} \left(-\mathbb{E}[\ln p(y_k,x_k)]_{q(x_k)} + \mathbb{E}[\ln q(x_k)]_{q(x_k)}\right) \tag{4.3}$$

Where the k-index represents the time-step associated with a specific time-instance. The state sequence x and output sequence y are compact notations:

$$x = [x_{k=1}x_{k=2}\dots x_{k=K}] \text{ and } y = [y_{k=1}y_{k=2}\dots y_{k=K}]$$

It is likely that both the state and output metric for any time-instance contain multiple state and output variable:

$$x_k = [x_1 x_2 \dots x_n]^T \text{ and } y_k = [y_1 y_2 \dots y_m]^T$$

where n represents the order of the Process and m represents the number of measured outputs of the Process. The order and number of inputs (p) and outputs are typically selected to be constant for the selected time period. The $\prod$ symbol part of equation 4.3 represents the product of all individual probability density: the joint probability density. Because of the use of the log function the product by definition becomes a summation [41]. The joint probability can be written as a function of the state Likelihood and Prior probabilities: $p(x,y) = p(y|x)p(x)$, which is known as the product rule.

For a particular time-step, the Free Energy equation becomes:

$$(\mathscr{F}_x)_k = -\mathbb{E}[\ln p(y_k|x_k)]_{q(x_k)} - \mathbb{E}[\ln p(x_k)]_{q(x_k)} + \mathbb{E}[\ln q(x_k)]_{q(x_k)} \tag{4.4}$$

In many cases, it is acceptable to assume Gaussian distributed probabilities. A Gaussian distribution is determined solely by the first and second statistical moment: the mean and variance. The mean is the expected value of a random variable: $\mu = \mathbb{E}[x]$, the variance is the expected value of the squared deviation from the mean of x: $\Sigma = \mathbb{E}[(x-\mu)^2]$. To specify the probabilities that are part of the Free Energy equation, the phrase Belief is introduced. A belief corresponds to a prediction made with the $y_{1:k-1}$ measurements that are available [47]. The state Likelihood and Prior Gaussian probabilities are a function of the belief mean: $\bar{\mu}$ and variance $\bar{\Sigma}$. Both the state Likelihood and Prior Gaussian probabilities are conditioned on the $y_{1:k-1}$ measurements that are available. This does not apply to the auxiliary distribution $q(x_k)$ which is optimized to be as close as possible to the Posterior probability $p(x_k|y_k)$.

The state space representation used to represent the dynamics of the generative Process as was presented in chapter 2 (equations 2.1 and 2.2) are used to further specify the means of the state and output variable. By taking the expectation over the state and output the belief state mean $\bar{\mu}_x$ and prediction of the output $\hat{y}_k$ are found. The noises are assumed to be zero mean (Wiener Process). The expectation over the Process and Measurement Noises are therefore equal to zero. The functions part of the state space representation $f(\cdot)$ and $g(\cdot)$ will be further specified later in the chapter. Currently, the state space representation is not limited to linear dynamical Processes.

$$\mathbb{E}[x_k] = \mathbb{E}[f(x_{k-1}, u_{k-1})] + \mathbb{E}[w_k] \qquad (\bar{\mu}_x)_k = \mathbb{E}[f(x_{k-1}, u_{k-1})] \qquad \text{Transition model} \tag{4.5}$$

$$\mathbb{E}[y_k] = \mathbb{E}[g(x_k)] + \mathbb{E}[z_k] \qquad \hat{y}_k = \mathbb{E}[g(x_k)] \qquad \text{Observation model} \tag{4.6}$$

The mean and variance that determine a multivariate Gaussian distribution of a random vector x are defined as a n-dimensional mean vector: $\mu = \mathbb{E}[x] = [\mathbb{E}[x_1], \mathbb{E}[x_2], \ldots, \mathbb{E}[x_n]]^T$ and $n \times n$ variance matrix: $\Sigma = \mathbb{E}[(x-\mu)(x-\mu)^T]$ [28]. The time-index of the means and variances are omitted such that a more clear notation results.

The probabilities that are part of the Free Energy equation are characterized -using the information provided- as:

$$q(x_k) \backsim \mathcal{N}(x_k|\mu_x, \Sigma_x) = \frac{1}{\sqrt{2\pi^n|\Sigma_x|}} \exp\left\{-\frac{1}{2}(x_k-\mu_x)^T\Sigma_x^{-1}(x_k-\mu_x)\right\}$$

$$p(y_k|x_k) \backsim \mathcal{N}(y_k|\mathbb{E}[g(x_k)], \bar{\Sigma}_y) = \frac{1}{\sqrt{2\pi^m|\bar{\Sigma}_y|}} \exp\left\{-\frac{1}{2}(y_k-\mathbb{E}[g(x_k)])^T\bar{\Sigma}_y^{-1}(y_k-\mathbb{E}[g(x_k)])\right\}$$

$$p(x_k) \backsim \mathcal{N}(x_k|\mathbb{E}[f(x_{k-1}, u_{k-1})], \bar{\Sigma}_x) = \frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}} \exp\left\{-\frac{1}{2}(x_k-\mathbb{E}[f(x_{k-1}, u_{k-1})])^T\bar{\Sigma}_x^{-1}(x_k-\mathbb{E}[f(x_{k-1}, u_{k-1})])\right\}$$

The characterizations of the probabilities are substituted into the Free Energy equation (equation 4.4). With logarithmic rules the Free Energy equation is simplified.

$$
\begin{aligned}
(\mathscr{F}_x)_k =& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^n|\Sigma_x|}}\exp\left\{-\frac{1}{2}(x_k-\mu_x)^T\Sigma_x^{-1}(x_k-\mu_x)\right\}\right)\right] - \\
& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^m|\bar{\Sigma}_y|}}\exp\left\{-\frac{1}{2}(y_k-\mathbb{E}[g(x_k)])^T\bar{\Sigma}_y^{-1}(y_k-\mathbb{E}[g(x_k)])\right\}\right)\right] - \\
& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}}\exp\left\{-\frac{1}{2}(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])^T\bar{\Sigma}_x^{-1}(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])\right\}\right)\right]
\end{aligned} \tag{4.7}
$$

$$
\begin{aligned}
(\mathscr{F}_x)_k =& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^n|\Sigma_x|}}\right)\right] - \mathbb{E}\left[\frac{1}{2}(x_k-\mu_x)^T\Sigma_x^{-1}(x_k-\mu_x)\right] - \\
& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^m|\bar{\Sigma}_y|}}\right)\right] - \mathbb{E}\left[\frac{1}{2}(y_k-\mathbb{E}[g(x_k)])^T\bar{\Sigma}_y^{-1}(y_k-\mathbb{E}[g(x_k)])\right] - \\
& \mathbb{E}\left[\ln\left(\frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}}\right)\right] - \mathbb{E}\left[\frac{1}{2}(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])^T\bar{\Sigma}_x^{-1}(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])\right]
\end{aligned} \tag{4.8}
$$

$$
\begin{aligned}
(\mathscr{F}_x)_k =& -\frac{1}{2}\mathbb{E}\left[\ln\left(|\Sigma_x|\right)\right] - \frac{1}{2}\mathbb{E}\left[(x_k-\mu_x)^T\Sigma_x^{-1}(x_k-\mu_x)\right] \\
& + \frac{1}{2}\mathbb{E}\left[\ln\left(|\bar{\Sigma}_y|\right)\right] + \frac{1}{2}\mathbb{E}\left[(y_k-\mathbb{E}[g(x_k)])^T\bar{\Sigma}_y^{-1}(y_k-\mathbb{E}[g(x_k)])\right] \\
& + \frac{1}{2}\mathbb{E}\left[\ln\left(|\bar{\Sigma}_x|\right)\right] + \frac{1}{2}\mathbb{E}\left[(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])^T\bar{\Sigma}_x^{-1}(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])\right] + \text{constants}
\end{aligned} \tag{4.9}
$$

The Free Energy Equation is further simplified by solving the second term of each row where the term $\frac{1}{2}\mathbb{E}\left[\{(x_k-\mu_x)^T\Sigma_x^{-1}(x_k-\mu_x)\}\right]$ reduces to a constant. The reduction of the remaining two terms is written underneath where the variance terms are left out in order to keep a clear notation.

$$
\begin{aligned}
&= \mathbb{E}[(y_k-\mathbb{E}[g(x_k)])^T(y_k-\mathbb{E}[g(x_k)])] \\
&= \mathbb{E}[y_k^Ty_k - y_k^T\mathbb{E}[g(x_k)] - \mathbb{E}[g(x_k)^T]y_k + \mathbb{E}[g(x_k)^T]\mathbb{E}[g(x_k)]] \\
&= \mathbb{E}[y_k^Ty_k] - \mathbb{E}[y_k^T]\mathbb{E}[g(x_k)] - \mathbb{E}[g(x_k)^T]\mathbb{E}[y_k] + \mathbb{E}[g(x_k)^T]\mathbb{E}[g(x_k)] \\
&= y_k^Ty_k - y_k^T\mathbb{E}[g(x_k)] - \mathbb{E}[g(x_k)^T]y_k + \mathbb{E}[g(x_k)^T]\mathbb{E}[g(x_k)] \\
&= (y_k-\mathbb{E}[g(x_k)])^T(y_k-\mathbb{E}[g(x_k)])
\end{aligned} \tag{4.10}
$$

$$
\begin{aligned}
&= \mathbb{E}[(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])^T(x_k-\mathbb{E}[f(x_{k-1},u_{k-1})])] \\
&= \mathbb{E}[x_k^Tx_k - x_k^T\mathbb{E}[f(x_{k-1},u_{k-1})] - x_k\mathbb{E}[f(x_{k-1},u_{k-1})^T] + \mathbb{E}[f(x_{k-1},u_{k-1})^T]\mathbb{E}[f(x_{k-1},u_{k-1})]] \\
&= \mathbb{E}[x_k^Tx_k] - \mathbb{E}[x_k^T]\mathbb{E}[f(x_{k-1},u_{k-1})] - \mathbb{E}[x_k]\mathbb{E}[f(x_{k-1},u_{k-1})^T] + \mathbb{E}[f(x_{k-1},u_{k-1})^T]\mathbb{E}[f(x_{k-1},u_{k-1})]] \\
&= (\mu_x-\bar{\mu}_x)^T(\mu_x-\bar{\mu}_x) + \mathbb{E}[x_k^Tx_k] - \mathbb{E}[x_k]^T\mathbb{E}[x_k]
\end{aligned} \tag{4.11}
$$

In case of one-dimensionality, the grey terms present in equation 4.11 are equal to the variance Σ_x. The term part of the Free Energy equation would be $\Sigma_x \bar{\Sigma}_x^{-1}$. In this chapter state estimation is considered meaning the Free Energy is evaluated with respect to the state. An estimation of the state is found when the Free Energy equation has a minimal value. The derivative of the Free Energy with respect to the state is taken. Both the constant terms and grey terms present in equation 4.11 will disappear and are therefore no longer taken into account. For the multi-dimensional case something similar applies but require additional algebra. The equations 4.10 and 4.11 are substituted back into the Free Energy equation 4.9. The Free Energy equation reduces to:

$$(\mathscr{F}_x)_k = -\frac{1}{2}\mathbb{E}\left[\ln\left(|\Sigma_x|\right)\right] + \frac{1}{2}\mathbb{E}\left[\ln\left(|\bar{\Sigma}_y|\right)\right] + \frac{1}{2}\mathbb{E}\left[\ln\left(|\bar{\Sigma}_x|\right)\right] + \frac{1}{2}\left\{(y_k - \mathbb{E}[g(x_k)])^T \bar{\Sigma}_y^{-1}(y_k - \mathbb{E}[g(x_k)])\right\} + \frac{1}{2}\left\{(\mu_x - \bar{\mu}_x)^T \bar{\Sigma}_x^{-1}(\mu_x - \bar{\mu}_x)\right\} \tag{4.12}$$

The remaining expectation $E[g(x_k)]$ is specified as:

$$\mathbb{E}[g(f(x_{k-1}, u_{k-1}))] = g(\mathbb{E}[f(x_{k-1}, u_{k-1})]) = g(\bar{\mu}_x)$$

The final form of the Free Energy equation with respect to the state is given as:

$$(\mathscr{F}_x)_k = \epsilon_y^T \bar{\Sigma}_y^{-1} \epsilon_y + \epsilon_x^T \bar{\Sigma}_x^{-1} \epsilon_x + \ln(\bar{\Sigma}_x \bar{\Sigma}_y) - \ln(\Sigma_x) \tag{4.13}$$

Where $\epsilon_y = y_k - g((\bar{\mu}_x)_k)$ and $\epsilon_x = (\mu_x)_k - (\bar{\mu}_x)_k$.

The most simple form of finding the minimum of the Free Energy equation with respect to the state is to find the state estimation for which the derivative of the Free Energy equation is equal to zero. Note that the state estimate μ_x ultimately is the best representation of the true state and not the belief state that represents an initial prediction. By minimization of the Free Energy, the divergence between the true Posterior distribution and Recognition Density is minimized as well.

$$\frac{\partial (\mathscr{F}_x)_k}{\partial \mu_x} = \frac{-\partial g((\bar{\mu}_x)_k)}{\partial \mu_x}(\bar{\Sigma}_y)^{-1}(y_k - g((\bar{\mu}_x)_k)) + (\bar{\Sigma}_x)^{-1}((\mu_x)_k - (\bar{\mu}_x)_k) = 0 \tag{4.14}$$

By a simple rearrangement of equation 4.14 a very familiar form is obtained. The following section of this chapter will address the relation between the derived equation 4.14 and the Kalman Filter algorithm and prove that the equality presented in equation 4.17 is true.

$$(\mu_x)_k = (\bar{\mu}_x)_k + \frac{\partial g((\bar{\mu}_x)_k)}{\partial \mu_x}(\bar{\Sigma}_x)(\bar{\Sigma}_y)^{-1}(y_k - g((\bar{\mu}_x)_k)) \tag{4.15}$$

$$(\mu_x)_k = (\bar{\mu}_x)_k + K_k(y_k - g((\bar{\mu}_x)_k)) \tag{4.16}$$

$$K_k = \frac{\partial g((\bar{\mu}_x)_k)}{\partial \mu_x}(\bar{\Sigma}_x)(\bar{\Sigma}_y)^{-1} = \frac{\partial g((\bar{\mu}_x)_k)}{\partial \mu_x}\text{var}(x_k x_k | y_{0:k-1})(\text{var}(y_k y_k | y_{0:k-1}))^{-1} \tag{4.17}$$

For the derivations the references [3], [4], [17] and [26] where regarded as useful.

4.1. THE RELATION OF FE STATE ESTIMATION WITH THE KALMAN FILTER ALGORITHM

In the Kalman Filter algorithm a linear Gaussian state space model is used. Thus far, in the derivations of the first part of this chapter a more generic state space model -applicable to nonlinear Processes- has been used. Many real-world processes are nonlinear but often can be approximated using linear models [31]. A linear state space representation with noise characteristics is represented as:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \qquad w_k \backsim \mathcal{N}(0, \Sigma_w) \tag{4.18}$$

$$y_k = Cx_k + z_k \qquad z_k \backsim \mathcal{N}(0, \Sigma_z) \tag{4.19}$$

Where:

$$\mathbb{E}\begin{bmatrix} w_k w_k^T & w_k z_k^T \\ z_k w_k^T & z_k z_k \end{bmatrix} = \begin{bmatrix} \Sigma_w & 0 \\ 0 & \Sigma_z \end{bmatrix}$$

To prove the equality presented in equation 4.17 to be true from now on linearity is assumed. Similar to the first part of this chapter the Kalman filter also makes use of the belief state estimate. The Kalman filter algorithm is split into two consecutive steps conducted in an iterative manner. In the prediction step, a state estimation is executed conditioned on the response up to, but not including, the current observation. In the correction step, this estimation is corrected such that the state estimation is conditioned upon all observations [46]. Equation 4.16 represent the correction step.

The state estimation according to the principles of Free Energy represented by equation 4.15 is a function of the belief state variance and belief output variance. To prove the equality presented in equation 4.17 to be true both of the belief variances need to be specified. To do the specification it is useful to specify the belief state mean. The expectation taken over the state up to the current observation is defined as:

$$\mathbb{E}[x_k]_{y_{1:k-1}} = (\bar{\mu}_x)_k = A(\mu_x)_{k-1} + Bu_{k-1} + \underbrace{\mathbb{E}[w_{k-1}]}_{=0} \tag{4.20}$$

The belief state variance is defined as:

$$\text{var}(x_k x_k | y_{1:k-1}) = \mathbb{E}[(x_k - (\bar{\mu}_x)_k)(x_k - (\bar{\mu}_x)_k)^T] = (\bar{\Sigma}_x)_k \tag{4.21}$$

By substitution of equations 4.18 and 4.20 the belief state variance is specified as:

$$\begin{aligned}
(\bar{\Sigma}_x)_k &= \mathbb{E}\left[(Ax_{k-1} + w_{k-1} - A(\bar{\mu}_x)_{k-1})(Ax_{k-1} + w_{k-1} - A(\bar{\mu}_x)_{k-1})^T\right] \\
&= \mathbb{E}\left[(A(x_{k-1} - (\bar{\mu}_x)_{k-1}) + w_{k-1})(A(x_{k-1} - (\bar{\mu}_x)_{k-1}) + w_{k-1})^T\right] \\
&= \mathbb{E}\left[(A(\epsilon_x)_{k-1} + w_{k-1})(A(\epsilon_x)_{k-1} + w_{k-1})^T\right] \\
&= A\mathbb{E}\left[(\epsilon_x)_{k-1}(\epsilon_x)_{k-1}^T\right] A^T + \mathbb{E}\left[w_{k-1} w_{k-1}^T\right]
\end{aligned}$$

$$(\bar{\Sigma}_x)_k = A(\Sigma_x)_{k-1} A^T + (\Sigma_w)_{k-1} \tag{4.22}$$

Where $(\Sigma_x)_k$ represents the state variance.

A similar approach applies for the specification of the belief output variance. To do the specification it is useful to specify the prediction of the output. The expectation taken over the output is defined as:

$$\mathbb{E}[y_k] = \hat{y}_k = \mathbb{E}[Cx_k + z_k] = C\mathbb{E}[x_k] + \underbrace{\mathbb{E}[z_k]}_{=0} \tag{4.23}$$

The belief output variance is defined as:

$$\text{var}(y_k y_k | y_{0:k}) = \mathbb{E}[(y_k - \hat{y}_k)(y_k - \hat{y}_k)^T] = (\bar{\Sigma}_y)_k \tag{4.24}$$

By substitution of equations 4.19 and 4.23 the belief state variance is specified as:

$$\begin{aligned}(\bar{\Sigma}_y)_k &= \mathbb{E}[(y_k - C(\bar{\mu}_x)_k)(y_k - (C\bar{\mu}_x)_k)^T] \\ &= \mathbb{E}[(C(x_k - (\bar{\mu}_x)_k) + z_k)(C(x_k - (\bar{\mu}_x)_k) + z_k)^T] \\ &= \mathbb{E}[C(x_k - (\bar{\mu}_x)_k)(x_k - (\bar{\mu}_x)_k)^T C^T] + \mathbb{E}[z_k z_k^T]\end{aligned}$$

$$(\bar{\Sigma}_y)_k = C(\bar{\Sigma}_x)_k C^T + (\Sigma_z)_k \tag{4.25}$$

In order to complete the derivation of the variances, the cross-variance is derived. This is not required to prove the equality presented in equation 4.17 because equation 4.15 is only a function of the auto variances.

The Cross covariance is derived as:

$$\begin{aligned}\text{cov}(y_k, x_k | y_{1:k}) &= \mathbb{E}[(y_k - C(\bar{\mu}_x)_k)(x_k - (\bar{\mu}_x)_k)^T)] \\ &= C(\bar{\Sigma}_x)_k\end{aligned}$$

The full auto and cross covariance matrix is specified as [26]:

$$\begin{bmatrix} x_k \\ y_k \end{bmatrix} \backsim \mathcal{N}\left(\begin{bmatrix} (\bar{\mu}_x)_k \\ C(\bar{\mu}_x)_k \end{bmatrix}, \begin{bmatrix} (\bar{\Sigma}_x)_k & (\bar{\Sigma}_x)_k C^T \\ C(\bar{\Sigma}_x)_k & C(\bar{\Sigma}_x)_k C^T + (\Sigma_z)_k \end{bmatrix}\right)$$

In case of a linearity equation 4.13 transforms into the following equation:

$$(\mathscr{F}_x)_k = (y_k - C(\bar{\mu}_x)_k)^T \bar{\Sigma}_y^{-1} (y_k - C(\bar{\mu}_x)_k) + ((\mu_x)_k - (\bar{\mu}_x)_k)^T \bar{\Sigma}_x^{-1} ((\mu_x)_k - (\bar{\mu}_x)_k) \tag{4.26}$$

By substitution of the belief variances into the Free Energy equation 4.26 the following extension results:

$$(\mathscr{F}_x)_k = (y_k - C(\bar{\mu}_x)_k)^T (C(\bar{\Sigma}_x)_k C^T + (\Sigma_z)_k)^{-1} (y_k - C(\bar{\mu}_x)_k) + ((\mu_x)_k - (\bar{\mu}_x)_k)^T \bar{\Sigma}_x^{-1} ((\mu_x)_k - (\bar{\mu}_x)_k) \tag{4.27}$$

The minimum of the Free Energy with respect to the state is again found by taking the derivative. In case of linearity equation 4.14 transforms into the following equation:

$$\frac{\partial (\mathscr{F}_x)_k}{\partial \mu} = -C^T (C(\bar{\Sigma}_x)_k C^T + (\Sigma_z)_k)^{-1} (y_k - C(\bar{\mu}_x)_k) + (\bar{\Sigma}_x)_k)^{-1} ((\mu_x)_k - (\bar{\mu}_x)_k) = 0 \tag{4.28}$$

By a simple rearrangement of equation 4.28 the equality presented in equation 4.17 has been proven correct. In case of linearity and with substitution of the belief variances equation 4.15 transforms into equation 4.29 which is the exact equivalent of the Kalman filter.

$$(\mu_x)_k = (\bar{\mu}_x)_k + \underbrace{(\bar{\Sigma}_x)_k C^T (C(\bar{\Sigma}_x)_k C^T + (\Sigma_z)_k)^{-1}}_{=K_k} (y_k - C(\bar{\mu}_x)_k) \tag{4.29}$$

4.2. DISCUSSION AND CONCLUSION

The equality presented in equation 4.17 has been proven correct. This is not particularly surprising although the proof is not straightforward (or trivial). In chapter 3 a statement is made that the Learning of the model part of the Free Energy principle without the incorporation of the concept of generalized motion, a conventionally generated input trajectory and a simple form of optimization to be in correspondence with the Variational EM algorithm. In almost all (Variational) EM algorithms the expectation step is assumed to compute Kalman estimates where the used references [4], [24], [31] and [41] are all examples off. Because the Kalman filter is so well-known explicit derivations such as is presented in this chapter is often skipped resulting in the (Variational) EM algorithm to be less comprehensible. The relation between the Free Energy principle and the Kalman filter is acknowledged by K. Friston although with the Continuous-Time Kalman-Bucy filter [27].

In the Kalman Filter algorithm, there in addition is a computation of the state variance Σ_x. A similar evaluation of minimization of the Free Energy but now with respect to the variance should lead to the same equivalence as was adopted with respect to the mean.

$$\frac{\partial (\mathcal{F}_x)_k}{\partial \Sigma_x} = 0 \tag{4.30}$$

In the chapters 6 and 7 a Free Energy Learning algorithm will be implemented. Although derived from the information construct Free Energy, it is not entirely fair to refer to a Free Energy Learning algorithm or similar to state estimation according to the principles of Free Energy. The state estimation according to the principles of Free Energy -in this case- will be an implementation of the Kalman Filter algorithm as the equivalence is proven in this chapter. Once the concept of generalized motion is incorporated, there will be a distinction. The formulation however remains distinguishable.

Following the ideas of the EM algorithm now that the state (and also input) is known the estimation of the model parameters should be simple. It should be capable to determine the model parameters by a simple linear regression problem. In the next chapter a similar approach is followed as done in this chapter. The Free Energy minimization problem will be evaluated with respect to the model parameters. At the end of the derivation, it shall turn out whether this indeed reduces to a simple linear regression problem.

5

THE FREE ENERGY PRINCIPLE: LEARNING THE MODEL MATRICES

This chapter will evaluate the second problem of equation 3.12 confirm to the Maximization step of the variational EM algorithm. Primarily, this chapter serves to explain the Free Energy principle in relation to the Learning of the model. A similar approach as the previous chapter 4 is followed. First the Free Energy with respect to the model parameter is specified. Second, the probabilities part of the specified Free Energy are characterized. And last, the Free Energy is minimized with respect to the model parameters. The model parameters are the matrices that model the generative process represented by the linear state space equations 4.18 and 4.19.

Remember, the Free Energy equation to be specified as [17]:

$$\mathscr{F} = -\mathbb{E}[\ln p(y,\vartheta)]_{q(\vartheta)} + \mathbb{E}[\ln q(\vartheta)]_{q(\vartheta)} \tag{5.1}$$

The causes set ϑ is partitioned into the state x, the model parameters $\theta = [A,B,C]$, the hyper parameters $\lambda = [\tau,\rho,\alpha,\beta,\gamma]$ and the input u. The model matrix D is often equal to zero and therefore is not taken into account. The symbols τ and ρ represent the noise precision's that belongs to the noise variances $\Sigma_w(\tau^{-1})$ and $\Sigma_z(\rho^{-1})$. The explicit specification of the noise precision's allows for the computation of the variances. Typically, the noise variances are specified by the engineers and given, causing a decrease in the adaptability of the application towards uncertain environments. To present an impression of the possibilities associated to Free Energy minimization the noise precision's are not yet omitted from the Free Energy equation. The same applies for the precision's associated to the variances of the model matrices $\Sigma_A(\alpha^{-1})$, $\Sigma_B(\beta^{-1})$ and $\Sigma_C(\gamma^{-1})$. By substitution of the causes set $\vartheta = \{x,A,\alpha,B,\beta,C,\gamma,\tau,\rho,u\}$ the Free Energy equation 5.1 is extended to equation 5.2. This equation 5.2 represents a more specified version of the equation 4.2 presented in chapter 4. The probability product rule $p(x,y) = p(x|y)p(y) = p(y|x)p(x)$ and logarithmic rule $\ln(ab) = \ln(a)+\ln(b)$ are used to further specify equation 5.2.

$$\mathscr{F} = -\mathbb{E}[\ln p(y,x,A,\alpha,B,\beta,C,\gamma,\tau,\rho,u)] + \mathbb{E}[\ln(q(x,A,\alpha,B,\beta,C,\gamma,\tau,\rho,u))] \tag{5.2}$$

$$\mathscr{F} = -\mathbb{E}[\ln p(y,x,A,\alpha,B,\beta,C,\gamma,\tau,\rho,u)] + \mathbb{E}[\ln(q(x)q(A)q(\alpha)q(B)q(\beta)q(C)q(\gamma)q(\tau)q(\rho)q(u))] \tag{5.3}$$

$$\begin{aligned}\mathscr{F} = &-\mathbb{E}[\ln p(y|x,C,\rho)p(x|A,B,u,\tau)p(A|\alpha)p(\alpha)p(B|\beta)p(\beta)p(\tau)p(C|\gamma)p(\gamma)p(\rho)p(u)] \\ &+\mathbb{E}[\ln(q(x)q(A)q(\alpha)q(B)q(\beta)q(C)q(\gamma)q(\tau)q(\rho)q(u))]\end{aligned} \tag{5.4}$$

$$\begin{aligned}\mathscr{F} = &-\mathbb{E}[\ln p(y|x,C,\rho)] - \mathbb{E}[\ln p(x|A,B,u,\tau)] - \mathbb{E}[\ln p(A|\alpha)] - \mathbb{E}[\ln p(\alpha)] - \mathbb{E}[\ln p(\beta)] - \mathbb{E}[\ln p(\tau)] \\ &-\mathbb{E}[\ln p(C|\gamma)] - \mathbb{E}[\ln p(\gamma)] - \mathbb{E}[\ln p(\rho)] - \mathbb{E}[\ln p(u)] + \mathbb{E}[\ln(q(x)] + \mathbb{E}[\ln q(A)] + \mathbb{E}[\ln q(\alpha)] \\ &+\mathbb{E}[\ln q(B)] + \mathbb{E}[\ln q(\beta)] + \mathbb{E}[\ln q(C)] + \mathbb{E}[\ln q(\gamma)] + \mathbb{E}[\ln q(\tau)] + \mathbb{E}[\ln q(\rho)] + \mathbb{E}[\ln q(u)]\end{aligned} \tag{5.5}$$

If every probability of the Free Energy equation 5.5 is characterized this full specification of the Free Energy can be used to compute every quantity of the causes set such that minimization of the Free Energy results. The probabilities that are a function of the hyper parameters represent a Prior probability distribution. It represents the Prior distributions over all of the variances. The precision's control the amplitude and smoothness of the random fluctuations [12] The main advantage of the Bayesian framework is that the inclusion of prior knowledge is coherent and, therefore, arises naturally as is seen in equations 5.2-5.5. The Prior distribution should be selected as a reflection of any prior beliefs (available knowledge beforehand) [5]. The inclusion of the hyper parameters could be advantageous but require additional research. The reference [4] can be consulted for an example of the inclusion of Prior probability distributions similar to what so far is presented. The Prior probability distributions are in the reference characterized by Gamma distributions. In case of the inclusion of the hyper parameters, the equations for the model matrix estimates will be a function of the hyper parameters. From now on, however, the hyper parameters of the model matrices will be neglected. The noise variances will be assumed given. A specification of the precision's ρ and τ is therefore not required. Without the hyper parameters the Free Energy equation 5.5 and with respect to the model matrices A, B and C reduces to:

$$\begin{aligned}\mathscr{F}_{A,B,C} = &-\mathbb{E}[\ln p(y|x,C,\bar{\Sigma}_y)] - \mathbb{E}[\ln p(x|A,B,u,\bar{\Sigma}_x)] \\ &+\mathbb{E}[\ln q(A)] + \mathbb{E}[\ln q(B)] + \mathbb{E}[\ln q(C)]\end{aligned} \tag{5.6}$$

The probabilities that are part of the Free Energy equation 5.6 are characterized. For the characterization of the recognition density a matrix Gaussian distribution notation is used [28].

$$p(y_k|x_k,C,\bar{\Sigma}_y) \backsim \mathcal{N}(y_k|Cx_k,\bar{\Sigma}_y) = \frac{1}{\sqrt{2\pi^m|\bar{\Sigma}_y|}}\exp\left\{-\frac{1}{2}(y_k - C(\bar{\mu}_x)_k)^T(\bar{\Sigma}_y)^{-1}(y_k - C(\bar{\mu}_x)_k)\right\}$$

$$p(x_k|A,B,u_{k-1},\bar{\Sigma}_x) \backsim \mathcal{N}(x_k|Ax_{k-1}+Bu_{k-1},\bar{\Sigma}_x) = \frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}}\exp\left\{-\frac{1}{2}(x_k - A(\mu_x)_{k-1} - Bu_{k-1})^T(\bar{\Sigma}_x)^{-1}(x_k - A(\mu_x)_{k-1} - Bu_{k-1})\right\}$$

$$q(A) \backsim \mathcal{N}(A|\mu_A,\Sigma_A) = \frac{1}{\sqrt{2\pi^n|\Sigma_A|}}\exp\left\{-\frac{1}{2}(\text{vec}(A) - \text{vec}(\mu_A))^T\Sigma_A^{-1}(\text{vec}(A) - \text{vec}(\mu_A))\right\}$$

$$q(B) \backsim \mathcal{N}(B|\mu_B,\Sigma_B) = \frac{1}{\sqrt{2\pi^n|\Sigma_B|}}\exp\left\{-\frac{1}{2}(\text{vec}(B) - \text{vec}(\mu_B))^T\Sigma_B^{-1}(\text{vec}(B) - \text{vec}(\mu_B))\right\}$$

$$q(C) \backsim \mathcal{N}(C|\mu_C,\Sigma_C) = \frac{1}{\sqrt{2\pi^n|\Sigma_C|}}\exp\left\{-\frac{1}{2}(\text{vec}(C) - \text{vec}(\mu_C))^T\Sigma_C^{-1}(\text{vec}(C) - \text{vec}(\mu_C))\right\}$$

The characterizations of the probabilities are substituted into the Free Energy equation 5.6 which after is further simplified.

$$
\begin{aligned}
\mathscr{F}_{A,B,C} = &-\mathbb{E}[\ln \frac{1}{\sqrt{2\pi^m|\bar{\Sigma}_y|}} \exp\left\{-\frac{1}{2}(y_k - C(\bar{\mu}_x)_k)^T(\bar{\Sigma}_y)^{-1}(y_k - C(\bar{\mu}_x)_k)\right\}] \\
&-\mathbb{E}[\ln \frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}} \exp\left\{-\frac{1}{2}(x_k - A(\mu_x)_{k-1} - Bu_{k-1})^T(\bar{\Sigma}_x)^{-1}(x_k - A(\mu_x)_{k-1} - Bu_{k-1})\right\}] \\
&+\mathbb{E}[\ln \frac{1}{\sqrt{2\pi^n|\Sigma_A|}} \exp\left\{-\frac{1}{2}(\text{vec}(A) - \text{vec}(\mu_A))^T\Sigma_A^{-1}(\text{vec}(A) - \text{vec}(\mu_A))\right\}] \\
&+\mathbb{E}[\ln \frac{1}{\sqrt{2\pi^n|\Sigma_B|}} \exp\left\{-\frac{1}{2}(\text{vec}(B) - \text{vec}(\mu_B))^T\Sigma_B^{-1}(\text{vec}(B) - \text{vec}(\mu_B))\right\}] \\
&+\mathbb{E}[\ln \frac{1}{\sqrt{2\pi^n|\Sigma_C|}} \exp\left\{-\frac{1}{2}(\text{vec}(C) - \text{vec}(\mu_C))^T\Sigma_C^{-1}(\text{vec}(C) - \text{vec}(\mu_C))\right\}]
\end{aligned}
$$

$$
\begin{aligned}
\mathscr{F}_{A,B,C} = &\frac{1}{2}\mathbb{E}[\ln|\bar{\Sigma}_y|] + \frac{1}{2}\mathbb{E}[\{(y_k - C(\bar{\mu}_x)_k)^T\bar{\Sigma}_y(y_k - C(\bar{\mu}_x)_k)\}]+ \\
&\frac{1}{2}\mathbb{E}[\ln|\bar{\Sigma}_x|] + \frac{1}{2}\mathbb{E}[\{(x_k - A(\mu_x)_{k-1} - Bu_{k-1})^T\bar{\Sigma}_x(x_k - A(\mu_x)_{k-1} - Bu_{k-1})\}]- \\
&\frac{1}{2}\mathbb{E}[\ln|\Sigma_A|] - \frac{1}{2}\mathbb{E}[\ln|\Sigma_B|] - \frac{1}{2}\mathbb{E}[\ln|\Sigma_C|] + \text{constants}
\end{aligned}
$$

The properties required to further reduce the Free Energy are:

$$
\begin{aligned}
\mathbb{E}[A] &= \mu_A & \mathbb{E}[\text{vec}(A)\text{vec}(A)^T] &= \mu_A\mu_A^T + \Sigma_A \\
\mathbb{E}[B] &= \mu_B & \mathbb{E}[\text{vec}(B)\text{vec}(B)^T] &= \mu_B\mu_B^T + \Sigma_B \\
\mathbb{E}[C] &= \mu_C & \mathbb{E}[\text{vec}(C)\text{vec}(C)^T] &= \mu_C\mu_C^T + \Sigma_C \\
\mathbb{E}[x] &= \mu_x & \mathbb{E}[xx^T] &= \mu_x\mu_x^T + \Sigma_x
\end{aligned}
$$

Because of the Mean-Field assumption: $q(\vartheta) = \Pi q(\vartheta_i)$ it is allowed to separately calculate the derivative with respect to every variable contained in the causes set. In the next section the estimations are derived for model matrix A: the state transition matrix and model matrix B: the control input matrix. The **F**ree **E**nergy minimization problems with respect to model matrix A and B will be evaluated. Both will be required for the chapters 6 and 7 where the derived equations will be implemented on two control problems. The model matrix C: the output matrix follows a similar derivation. For simplicity it is assumed that this matrix is equal to identity and therefore this matrix does not have a great share in the implementations. The derivation of this matrix is trivial after the derivations of the A and B matrix and therefore will not be mentioned any further.

5.1. THE EVALUATION OF FE MINIMIZATION WITH RESPECT TO THE A-MATRIX

The Free Energy equation with respect to the A-matrix contains the following elements:

$$\mathscr{F}_A = -\mathbb{E}[\ln p(x|A,B,\bar{\Sigma}_x)] + \mathbb{E}[\ln(q(A)] \tag{5.7}$$

Which expands into the following Free Energy equation once the Gaussian probabilities are substituted:

$$\begin{aligned}\mathscr{F}_A = &-\mathbb{E}[\ln\left(\frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}}\right)] + \frac{1}{2}\mathbb{E}[(x_k - A\mathbb{E}[x_{k-1}] - Bu_{k-1})^T\bar{\Sigma}_x^{-1}(x_k - A\mathbb{E}[x_{k-1}] - Bu_{k-1})] \\ &+\mathbb{E}[\ln(\frac{1}{\sqrt{2\pi|\Sigma_A|}})] - \frac{1}{2}\mathbb{E}[(\text{vec}(A) - \text{vec}(\mu_A))^T\Sigma_A^{-1}(\text{vec}(A) - \text{vec}(\mu_A))]\end{aligned} \tag{5.8}$$

Useful relations are once again the characterizations of the two statistical moments that specify the Gaussian probability distribution the mean and variance [28]:

$$\mathbb{E}[A] = (\mathbb{E}[A_{ij}]) = \mu_A \tag{5.9}$$

$$\text{var}[A] = \mathbb{E}[\text{vec}(A - \mathbb{E}[A])\text{vec}(A - \mathbb{E}[A])^T] = \mathbb{E}[\text{vec}(A)\text{vec}(A)^T] - \mu_A\mu_A^T \tag{5.10}$$

Where $A \in \mathbb{R}^{n\times n}$, $\mu_A \in \mathbb{R}^{n\times n}$, $\Sigma_A \in \mathbb{R}^{n^2\times n^2}$ and vec(A) is the vectorization of matrix A equal to concatenation of its columns:

$$\text{vec}(A) = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{nn} \end{bmatrix} \tag{5.11}$$

With the two statistical moments μ_A and Σ_A an update equation for the estimation of model matrix A is derived. The minimum of the Free Energy equation with respect to model matrix A is found by calculation of the derivative with respect to: μ_A and Σ_A and setting both derivatives to zero. In order to achieve a clear notation the expectation from now on is noted as $\langle\rangle$ instead of $\mathbb{E}[]$. The Free Energy with respect to model matrix A is:

$$\begin{aligned}\mathscr{F}_A = &-\langle x_k^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle - \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}x_k\rangle + \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle + \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}Bu_{k-1}\rangle \\ &+\langle(Bu_{k-1})^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle - \ln(|\Sigma_A|) - \langle(\text{vec}(A) - \text{vec}(\mu_A))^T\Sigma_A^{-1}(\text{vec}(A) - \text{vec}(\mu_A))\rangle + \text{constants}\end{aligned} \tag{5.12}$$

Where the term $\langle(\text{vec}(A) - \text{vec}(\mu_A))^T\Sigma_A^{-1}(\text{vec}(A) - \text{vec}(\mu_A))\rangle$ reduces to a constant once the expectation is solved and therefore can be omitted from the Free Energy equation.

$$\begin{aligned}\mathscr{F}_A = &-\langle x_k^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle - \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}x_k\rangle + \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle \\ &+\langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}Bu_{k-1}\rangle + \langle(Bu_{k-1})^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle - \ln(|\Sigma_A|) + \text{constants}\end{aligned} \tag{5.13}$$

Because the calculated derivatives are with respect to μ_A and Σ_A the Free Energy equation has to be a function of both of these statistical moments. The expectations that are still part of the Free Energy equation have to be solved, where the definitions of both the mean $\langle A\rangle = \mu_A$ and variance $\Sigma_A = \langle\text{vec}(A)\text{vec}(A)^T\rangle - \mu_A\mu_A^T$ and therefore $\langle\text{vec}(A)\text{vec}(A)^T\rangle = \Sigma_A + \text{vec}(\mu_A)\text{vec}(\mu_A)^T$ are used. The Free Energy equation contains one difficulty $\langle x_{k-1}^T A^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle$.

A relation between A^TA, AA^T, $\text{vec}(A)\text{vec}(A)^T$ and $\bar{\Sigma}_x^{-1}$ has to be found. The first useful relation is:

$$x^TAx = \text{tr}(Axx^T) = \text{tr}(xx^TA) \tag{5.14}$$

$$(Ax)^T\bar{\Sigma}_x^{-1}Ax = \text{tr}((Ax)(Ax)^T\bar{\Sigma}_x^{-1}) \tag{5.15}$$

What remains is the relation between AA^T and $\text{vec}(A)\text{vec}(A)^T$. It turns out that AA^T is the summation of the block diagonal of $\text{vec}(A)\text{vec}(A)^T$. The term $\text{vec}(A)\text{vec}(A)^T$ is defined as:

$$\text{vec}(A)\text{vec}(A)^T = \begin{bmatrix} \begin{bmatrix}\text{cell-array}\{1,1\}\\ n\times n\end{bmatrix} & \begin{bmatrix}\text{cell-array}\{1,2\}\\ n\times n\end{bmatrix} & \cdots & \begin{bmatrix}\text{cell-array}\{1,n\}\\ n\times n\end{bmatrix} \\ \begin{bmatrix}\text{cell-array}\{2,1\}\\ n\times n\end{bmatrix} & \begin{bmatrix}\text{cell-array}\{2,2\}\\ n\times n\end{bmatrix} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix}\text{cell-array}\{n,1\}\\ n\times n\end{bmatrix} & \begin{bmatrix}\text{cell-array}\{n,2\}\\ n\times n\end{bmatrix} & \cdots & \begin{bmatrix}\text{cell-array}\{n,n\}\\ n\times n\end{bmatrix} \end{bmatrix} \tag{5.16}$$

Then:

$$\begin{aligned} AA^T &= \text{vec}(A)\text{vec}(A)^T\{1,1\} + \text{vec}(A)\text{vec}(A)^T\{2,2\} + \ldots + \text{vec}(A)\text{vec}(A)^T\{n,n\} \\ &= \sum_{n=1}^{n}\text{vec}(A)\text{vec}(A)^T\{n,n\} \end{aligned} \tag{5.17}$$

Using this information, the problematic term can be written as a function of μ_A and Σ_A:

$$\begin{aligned} \langle x_{k-1}^TA^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle &= \langle x_{k-1}\rangle^T\langle\text{tr}(AA^T\bar{\Sigma}_x^{-1})\rangle\langle x_{k-1}\rangle = \langle x_{k-1}\rangle^T\text{tr}(\sum_{n=1}^{n}\langle\text{vec}(A)\text{vec}(A)^T\{n,n\}\rangle)\bar{\Sigma}_x^{-1})\langle x_{k-1}\rangle \\ &= \langle x_{k-1}\rangle^T\text{tr}(\sum_{n=1}^{n}(\Sigma_A + \text{vec}(\mu_A)\text{vec}(\mu_A)^T)\{n,n\}\bar{\Sigma}_x^{-1})\langle x_{k-1}\rangle \end{aligned} \tag{5.18}$$

A point of attention is the independence of the expectations taken over the state-variable. The process of learning the model matrices corresponds to the Maximization step part of the EM algorithm. The state variable are assumed fixed during execution of this Maximization step. It should, therefore, be allowed to take the expectations separately. The double $\langle x_{k-1}^Tx_{k-1}\rangle = \langle x_{k-1}^T\rangle\langle x_{k-1}\rangle$ term then does not require further rearrangement actions.

The Free Energy equation 5.13 translates into:

$$\begin{aligned} \mathscr{F}_A = &-\langle x_k\rangle^T\bar{\Sigma}_x^{-1}\mu_A\langle x_{k-1}\rangle - \langle x_{k-1}\rangle^T\mu_A^T\bar{\Sigma}_x^{-1}\langle x_k\rangle + \langle x_{k-1}\rangle^T\text{tr}(\sum_{n=1}^{n}(\Sigma_A + \text{vec}(\mu_A)\text{vec}(\mu_A)^T)\{n,n\}\bar{\Sigma}_x^{-1})\langle x_{k-1}\rangle \\ &+ \langle x_{k-1}^T\rangle\mu_A^T\bar{\Sigma}_x^{-1}\langle B\rangle u_{k-1} + u_{k-1}^T\langle B\rangle^T\bar{\Sigma}_x^{-1}\mu_A\langle x_{k-1}\rangle - \ln(|\Sigma_A|) \end{aligned} \tag{5.19}$$

In order to find the update equations the following derivatives are calculated: $\frac{\partial\mathscr{F}_A}{\partial\Sigma_A}$ and $\frac{\partial\mathscr{F}_A}{\partial\mu_A}$.

The absolute value of a matrix is the determinant of the matrix. For the derivative of the log determinant, the following relation is used:

$$\frac{\partial}{\partial\Sigma_A}\ln\det(\Sigma_A) = (\Sigma_A)^{-T}$$

The variance update equation is derived as follows:

$$\frac{\partial \mathcal{F}_A}{\partial \Sigma_A} = \langle x_{k-1} \rangle^T \langle x_{k-1} \rangle \begin{bmatrix} \bar{\Sigma}_x^{-1} & 0 & \dots & 0 \\ 0 & \bar{\Sigma}_x^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \bar{\Sigma}_x^{-1} \end{bmatrix} - (\Sigma_A)^{-T} = 0$$

And, therefore:

$$\Sigma_A = \left(\begin{bmatrix} \bar{\Sigma}_x^{-1} & 0 & \dots & 0 \\ 0 & \bar{\Sigma}_x^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \bar{\Sigma}_x^{-1} \end{bmatrix}^T \langle x_{k-1} \rangle \langle x_{k-1} \rangle^T \right)^{-1} \tag{5.20}$$

The mean update equation is derived as follows:

$$\begin{aligned} \frac{\partial \mathcal{F}_A}{\partial \mu_A} = &- (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \langle x_k \rangle \langle x_{k-1} \rangle^T + (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \langle B \rangle u_{k-1} \langle x_{k-1} \rangle^T \\ &+ \langle x_{k-1} \rangle^T \langle x_{k-1} \rangle (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \mu_A = 0 \end{aligned} \tag{5.21}$$

By a simple rearrangement the equation 5.21is transformed into:

$$\begin{aligned} \mu_A &= \left((\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \langle x_k \rangle \langle x_{k-1} \rangle^T - (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \langle B \rangle u_{k-1} \langle x_{k-1} \rangle^T \right) \left(\langle x_{k-1} \rangle^T \langle x_{k-1} \rangle (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T}) \right)^{-1} \\ &= \left(\langle x_k \rangle \langle x_{k-1} \rangle^T - \langle B \rangle u_{k-1} \langle x_{k-1} \rangle^T \right) \left(\langle x_{k-1} \rangle^T \langle x_{k-1} \rangle \right)^{-1} \end{aligned} \tag{5.22}$$

When the product term is incorporated back into both the mean and variance update equation the following final equations result:

The variance:

$$(\Sigma_A)_k = \left(\begin{bmatrix} (\bar{\Sigma}_x^{-1})_{k-1} & 0 & \dots & 0 \\ 0 & (\bar{\Sigma}_x^{-1})_{k-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & (\bar{\Sigma}_x^{-1})_{k-1} \end{bmatrix}^T \prod_{k=1}^{K} \langle x_{k-1} \rangle \langle x_{k-1} \rangle^T \right)^{-1} \tag{5.23}$$

The mean:

$$\mu_A = \left(\prod_{k=1}^{K} \langle x_k \rangle \langle x_{k-1} \rangle^T - \prod_{k=1}^{K} \langle B \rangle u_{k-1} \langle x_{k-1} \rangle^T \right) \left(\prod_{k=1}^{K} \langle x_{k-1} \rangle^T \langle x_{k-1} \rangle \right)^{-1} \tag{5.24}$$

5.2. THE EVALUATION OF FE MINIMIZATION WITH RESPECT TO THE B-MATRIX

The Free Energy equation with respect to the B-matrix contains the following elements:

$$\mathscr{F}_B = -\mathbb{E}[\ln p(x|A,B,\bar{\Sigma}_x)] + \mathbb{E}[\ln(q(B)] \tag{5.25}$$

Which expands into the following Free Energy equation once the Gaussian probabilities are substituted:

$$\begin{aligned}\mathscr{F}_B = &-\mathbb{E}[\ln\left(\frac{1}{\sqrt{2\pi^n|\bar{\Sigma}_x|}}\right)] + \frac{1}{2}\mathbb{E}[(x_k - A\mathbb{E}[x_{k-1}] - Bu_{k-1})^T\bar{\Sigma}_x^{-1}(x_k - A\mathbb{E}[x_{k-1}] - Bu_{k-1})] \\ &+ \mathbb{E}[\ln(\frac{1}{\sqrt{2\pi|\Sigma_B|}})] - \frac{1}{2}\mathbb{E}[(\text{vec}(B) - \text{vec}(\mu_B))^T\Sigma_B^{-1}(\text{vec}(B) - \text{vec}(\mu_B))]\end{aligned} \tag{5.26}$$

Useful relations are once again the characterizations of the two statistical moments that specify the Gaussian probability distribution, the mean and variance:

$$\mathbb{E}[B] = (\mathbb{E}[B_{ij}]) = \mu_B \tag{5.27}$$

$$\text{var}[B] = \mathbb{E}[\text{vec}(B - \mathbb{E}[B])\text{vec}(B - \mathbb{E}[B])^T] = \mathbb{E}[\text{vec}(B)\text{vec}(B)^T] - \mu_B\mu_B^T \tag{5.28}$$

Where $B \in \mathbb{R}^{n\times p}$, $\mu_B \in \mathbb{R}^{n\times p}$, $\Sigma_B \in \mathbb{R}^{np\times np}$

In order to achieve a clear notation the expectation, from now on, is noted as $\langle\rangle$ instead of $\mathbb{E}[]$. The term $\langle(\text{vec}(B)-\text{vec}(\mu_B))^T\Sigma_B^{-1}(\text{vec}(B)-\text{vec}(\mu_B))\rangle$ once the expectation is solved reduces to a constant and therefore can be omitted. The Free Energy equation reduces to:

$$\begin{aligned}\mathscr{F}_B = &-\langle x_k^T\bar{\Sigma}_x^{-1}Bu_{k-1}\rangle + \langle(Ax_{k-1})^T\bar{\Sigma}_x^{-1}Bu_{k-1}\rangle - \langle(Bu_{k-1})^T\bar{\Sigma}_x^{-1}x_k\rangle + \langle(Bu_{k-1})^T\bar{\Sigma}_x^{-1}Ax_{k-1}\rangle \\ &+ \langle(Bu_{k-1})^T\bar{\Sigma}_x^{-1}Bu_{k-1}\rangle - \ln(|\Sigma_B|) + \text{constants}\end{aligned} \tag{5.29}$$

In a similar approach as done with the derivation of the model-matrix: A, the Free Energy equation is written as function of the statistical moments: μ_B and Σ_B.

$$\begin{aligned}\mathscr{F}_B = &-\langle x_k\rangle^T\bar{\Sigma}_x^{-1}\mu_B u_{k-1} + \langle x_{k-1}\rangle^T\langle A\rangle^T\bar{\Sigma}_x^{-1}\mu_B u_{k-1} - u_{k-1}^T\mu_B^T\bar{\Sigma}_x^{-1}\langle x_k\rangle + u_{k-1}^T\mu_B^T\bar{\Sigma}_x^{-1}\langle A\rangle\langle x_{k-1}\rangle \\ &+ u_{k-1}^T\text{tr}(\sum(\Sigma_B + \text{vec}(\mu_B)\text{vec}(\mu_B)^T)\{n,p\}\bar{\Sigma}_x^{-1})u_{k-1} - \ln(|\Sigma_B|) + \text{constants}\end{aligned} \tag{5.30}$$

In order to find the update equations the following derivatives are calculated: $\frac{\partial\mathscr{F}_B}{\partial\Sigma_B}$ and $\frac{\partial\mathscr{F}_B}{\partial\mu_B}$.

The variance update equation is derived as follows:

$$\frac{\partial\mathscr{F}_B}{\partial\Sigma_B} = \langle u_{k-1}\rangle^T\langle u_{k-1}\rangle\begin{bmatrix}\bar{\Sigma}_x^{-1} & 0 & \dots & 0\\ 0 & \bar{\Sigma}_x^{-1} & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \bar{\Sigma}_x^{-1}\end{bmatrix} + (\Sigma_B)^{-T} = 0$$

And, therefore:

$$\Sigma_B = \left(\begin{bmatrix}\bar{\Sigma}_x^{-1} & 0 & \dots & 0\\ 0 & \bar{\Sigma}_x^{-1} & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \bar{\Sigma}_x^{-1}\end{bmatrix}^T\langle u_{k-1}\rangle\langle u_{k-1}\rangle^T\right)^{-1} \tag{5.31}$$

The mean update equation is derived as follows:

$$\begin{aligned}\frac{\partial \mathcal{F}_B}{\partial \mu_B} = &-(\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\langle x_k\rangle\langle u_{k-1}\rangle^T + (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\langle A\rangle\langle x_{k-1}\rangle\langle u_{k-1}\rangle^T \\ &+ \langle u_{k-1}\rangle^T\langle u_{k-1}\rangle(\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\mu_B = 0\end{aligned} \tag{5.32}$$

And, therefore:

$$\begin{aligned}\mu_B &= \left((\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\langle x_k\rangle\langle u_{k-1}\rangle^T - (\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\langle A\rangle\langle x_{k-1}\rangle\langle u_{k-1}\rangle^T\right)\left(\langle u_{k-1}\rangle^T\langle u_{k-1}\rangle(\bar{\Sigma}_x^{-1} + \bar{\Sigma}_x^{-T})\right)^{-1} \\ &= \left(\langle x_k\rangle\langle u_{k-1}\rangle^T - \langle A\rangle\langle x_{k-1}\rangle\langle u_{k-1}\rangle^T\right)\left(\langle u_{k-1}\rangle^T\langle u_{k-1}\rangle\right)^{-1}\end{aligned} \tag{5.33}$$

When the product term is incorporated back into both the mean and variance update equation the following final equations result:

The variance:

$$(\Sigma_B)_k = \left(\begin{bmatrix}(\bar{\Sigma}_x^{-1})_{k-1} & 0 & \dots & 0 \\ 0 & (\bar{\Sigma}_x^{-1})_{k-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & (\bar{\Sigma}_x^{-1})_{k-1}\end{bmatrix}^T \prod_{k=1}^{K}\langle u_{k-1}\rangle\langle u_{k-1}\rangle^T\right)^{-1} \tag{5.34}$$

The mean:

$$\mu_B = \left(\prod_{k=1}^{K}\langle x_k\rangle\langle u_{k-1}\rangle^T - \prod_{k=1}^{K}\langle A\rangle x_{k-1}\langle u_{k-1}\rangle^T\right)\left(\prod_{k=1}^{K}\langle u_{k-1}\rangle^T\langle u_{k-1}\rangle\right)^{-1} \tag{5.35}$$

5.3. DISCUSSION AND CONCLUSION

According to the EM algorithm the model matrix estimations can be found by a simple linear regression problem once the state (and input) are known. The linear regression is gives as [41]:

$$\underbrace{\begin{bmatrix}\hat{A} & \hat{B}\\ \hat{C} & \hat{D}\end{bmatrix}}_{\Gamma} = \max_{\Gamma} \sum_{k=1}^{K} || \begin{bmatrix}x_{k+1}\\ y_k\end{bmatrix} - \begin{bmatrix}\hat{A} & \hat{B}\\ \hat{C} & \hat{D}\end{bmatrix} \begin{bmatrix}x_k\\ u_k\end{bmatrix} ||^2 \tag{5.36}$$

The solution of the linear regression is given as [36]:

$$\begin{bmatrix}\hat{A} & \hat{B}\\ \hat{C} & \hat{D}\end{bmatrix} = \sum_{k=1}^{K} \begin{bmatrix}x_{k+1}x_k^T & x_{k+1}u_k^T\\ y_k x_k^T & y_k u_k^T\end{bmatrix} \left(\begin{bmatrix}x_k x_k^T & x_k u_k^T\\ u_k x_k^T & u_k u_k^T\end{bmatrix} \right)^{-1} \tag{5.37}$$

Which indeed is equivalent to the derived equations 5.24 and 5.35 in this chapter. The solutions found correspond to a closed form solution. The derivations presented in this chapter 5 and also chapter 4 are a product of improvement while being implemented on the Control problem presented in the chapters 6 and 7. The equivalence of the state estimation according to the principles of Free Energy was identified quite earlier during the research that constituted this thesis. The equivalence presented in this chapter 5 was identified in the final stages of the research that constituted this thesis. This is unfortunate because this identification led to the conclusion for linear dynamical models the Free Energy Learning algorithm with a Wiener assumption on the noise to be equivalent to the regular EM algorithm. Some nice papers where found which could have significantly improve the implemented algorithm. Although derived from the information construct Free Energy, it not is entirely fair to refer to a Free Energy Learning algorithm or similar to model learning according to the principles of Free Energy.

6

A One-Dimensional problem

6.1. The problem: a Force driven Mass

As a first illustration of the Learning process of a Linear Dynamical state space model according to the Free Energy principle, the following "simple" one-dimensional problem is considered:

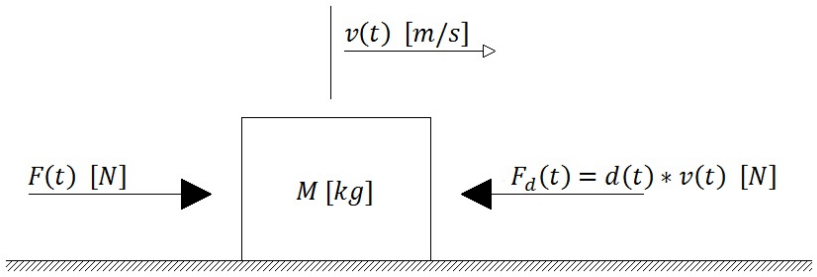


Figure 6.1: The True Process; a Force driven Mass

Where M represents the Mass, $F(t)$ represents the actuator Force, $v(t)$ represents the velocity of the Mass and $F_d(t)$ represents the drag Force. The drag Force is the measure of resistance encountered by the Mass. The drag Force equation is given as:

$$F_d(t) = \frac{1}{2}\rho v(t)^2 C_d A = d(t)v(t) \tag{6.1}$$

ρ = Air density [kg/m^3] $\quad$ $v(t)$ = velocity [m/s]

C_d = Drag-coefficient [] $\quad$ A = Reference area [m^2]

The problem is considered to be a very simple representation of a car. On average, the parameters of the problem based on parameters associated with a car, are:

Parameter	Value	Unit	Description
M	1000	[kg]	Mass of the car
A	3	[m^2]	Front Area
C_d	1/3	[]	Average car drag-coefficient
ρ	1.3	[kg/m^3]	Air density
d	$\approx$ 10	[Ns/m]	Coefficient of resistance belonging to a $\approx$ velocity of 50 km/h

Table 6.1: The parameters associated with the Force driven Mass problem

In order to maintain simplicity the coefficient of resistance d is assumed to remain a constant. In reality, the coefficient of resistance is dependent upon the fluctuations of the velocity. This sort of nonlinearity is very interesting but also complicated and is therefore part of future research.

Using Newton's Second Law: $\sum F(t) = M \cdot a(t)$ the Continuous-Time model that represents the one-dimensional problem is derived:

$$\underbrace{F(t) - F_d}_{\sum F} = \underbrace{m \cdot \dot{v}(t)}_{m \cdot a(t)} \tag{6.2}$$

By rearrangement the Continuous-Time model is formulated into a state space representation:

$$\dot{v}(t) = \frac{1}{m}(F(t) - d(t) \cdot v(t)) \qquad \dot{v}(t) = -\frac{d}{m}v(t) + \frac{1}{m}F(t) \tag{6.3}$$

Where:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \qquad \dot{x}(t) = \dot{v}(t) \qquad x(t) = v(t)$$
$$y(t) = C_c x(t) \qquad u(t) = F(t) \qquad y(t) = v(t)$$

$$A_c = -\frac{d}{m} \qquad B_c = \frac{1}{m} \qquad C_c = 1$$

The True Process Response is simulated as the Response to the state space representation perturbed by a random Process and Measurement Noise. The variance of the Measurement Noise is selected to be $\Sigma_w = 0.01$ and the variance of the Process Noise is selected to be $\Sigma_z = 0.01$. These noise levels correspond to an average variance of around 10% in the Process Response which is fairly high. However, the Learning and control algorithm should be capable of coping with these noise levels because of the simplicity of the considered problem. The state space Process has the following representation:

$$\dot{x}(t) = A_c x(t) + B_c u(t) + w(t) \qquad w(t) \backsim \mathcal{N}(0, \Sigma_w) \tag{6.4}$$
$$y(t) = C_c x(t) + z(t) \qquad z(t) \backsim \mathcal{N}(0, \Sigma_z) \tag{6.5}$$

The difference between the Process and model is solely due to the added noise component. This leads to the state of the Process to be different from the state of the model. The model and Process matrices A_c, B_c and C_c are equivalent in this case. In real set-ups this is obviously not the case. While the Free Energy Learning algorithm should expectantly be capable of dealing with model uncertainties:

$$\underbrace{A}_{Process} = \underbrace{\hat{A} + \Delta A}_{model}, \quad \underbrace{B}_{Process} = \underbrace{\hat{B} + \Delta B}_{model} \text{ and } \underbrace{C}_{Process} = \underbrace{\hat{C} + \Delta C}_{model}$$

The simulation of these model uncertainties is less straightforward. A More efficient approach is to first implement the Learning algorithm without the presence of model uncertainties and then followed by validating the Learning algorithm implementation against True Process behavior where model uncertainties are present.

To generate a useful Process Response, the simulation of the True Process is placed in a controlled environment. The Process Response to an arbitrary control input trajectory for example a Step, is highly unstable. While stability is not a requirement for the Learning algorithm to work it does lead to a more adequate basis to validate its performance. Furthermore, the complete Free Energy principle suggests that the Learning process is performed in a controlled environment. How precisely a control signal is generated according to the principles of Free Energy is not yet specified and is therefore beyond the scope of this thesis. As mentioned in the chapters 4 and 5 a Model Predictive Control Design is a sufficient alternative used to compute a trajectory of future manipulated variables u by optimizing the future behavior of the Process output Response y [52]. In contrast to the Free Energy principle -where beliefs generated by the brain serve as a Reference- in the Model Predictive Control Design an external Reference signal is provided. The desired Set-Point velocity is selected to be 50[km/h] $\approx$ 15[m/s].

6.2. THE MODEL PREDICTIVE CONTROL DESIGN

The Continuous-Time model is discretized to facilitate the Analog-to-Digital conversion: $x(t) \leftrightarrows x_k$. This process is known as Sampling and indicates the process of taking a small part or quantity of a Continuous-Time signal as a sample for testing and analysis. Sampling is a fundamental property of computer-controlled system because of the Discrete-Time nature of Digital computers. From Shannon's Theorem it is known that the process of Sampling has to be performed at a frequency that is at least twice the Bandwidth of interest [2]. The Bandwidth of the process is $\omega_b = 0.01$ [rad/s]. The sample time selected is dt= 0.1 [s]. The Discrete-Time state space model has the following representation:

$$x_{k+1} = A_d x_k + B_d u_k \tag{6.6}$$

$$y_k = C_d x_k \tag{6.7}$$

A **M**odel **P**redictive **C**ontrol (MPC) Design is based on the mathematical model of the True Process. A detailed Process model is required to realize a robust high-performance Process operation. There are multiple approaches to the Design of the Model Predictive Controller. Hence the MPC Design is a methodology rather than a single technique [49]. A standard form of the MPC Design is implemented. The Control Law is determined from minimization of a cost function which is selected as a measure of predictive performance. To ensure that the minimum achievable cost is $J = 0$ and to guarantee offset free control; the state space model is embedded with an Integrator. This necessarily requires the increment of the control signal rather than control signal itself [40]. The increments of the state variable x_k and input variable u_k, are :

$$\Delta x_k = x_k - x_{k-1} \qquad\qquad \Delta u_k = u_k - u_{k-1}$$

To gain the state space model embedded with an integrator the increments are incorporated as: $\Delta x_{k+1} = A_d \Delta x_k + B_d \Delta u_k$. To link the output to the increment of the state variable, the state is reformulated as $(x_a)_k = \begin{bmatrix} \Delta x_k & y_k \end{bmatrix}^T$. The state space model that result is known as the Augmented state space model which is a suitable state space model to use in the Design of Predictive Control:

$$\underbrace{\begin{bmatrix} \Delta x_{k+1} \\ y_{k+1} \end{bmatrix}}_{(x_a)_{k+1}} = \underbrace{\begin{bmatrix} A_d & 0 \\ C_d A_d & 1 \end{bmatrix}}_{A_a} \underbrace{\begin{bmatrix} \Delta x_k \\ y_k \end{bmatrix}}_{(x_a)_k} + \underbrace{\begin{bmatrix} B_d \\ C_d B_d \end{bmatrix}}_{B_a} \Delta u_k \tag{6.8}$$

$$y_k = \underbrace{\begin{bmatrix} 0 & 1 \end{bmatrix}}_{C_a} \begin{bmatrix} \Delta x_k \\ y_k \end{bmatrix} \tag{6.9}$$

This Augmented state space model serves two purposes in the MPC Design. First, the model is used to gain a prediction of the expected future Process output behavior. On the basis of inputs and known disturbances applied to the Process in the past. Second, the model is used to gain the next Process input that minimizes a predetermined Cost Function. Predictive Control follows the Receding Horizon principle. The prediction is described within an Optimization Window. Predictive Control considers predicted behavior over some Horizon into the future. At each successive Sampling Instant the Predictive Control Design predicts one sample further into the future. The Horizon selected for prediction, the Prediction Horizon N_p should include all significant dynamics [40]. Next to a Prediction Horizon there is a Control Horizon N_c, which is always less than the Prediction Horizon. The Control Horizon dictates the number of parameters used to capture the future control input. The incorporation of the Control and Prediction Horizon yields:

$$Y = \begin{bmatrix} y_{k+1|k} & y_{k+2|k} & \dots & y_{k+N_p|k} \end{bmatrix} \tag{6.10}$$

$$\Delta U = \begin{bmatrix} \Delta u_k & \Delta u_{k+1} & \dots & \Delta u_{k+N_c-1} \end{bmatrix} \tag{6.11}$$

Which are related through the following equation 6.12:

$$Y = F(x_a)_k + \Phi \Delta U \tag{6.12}$$

Where:

$$F = \begin{bmatrix} C_a A_a \\ C_a A_a^2 \\ \vdots \\ C_a A_a^{n_p} \end{bmatrix} \qquad \Phi = \begin{bmatrix} C_a B_a & 0 & \dots & 0 \\ C_a A_a B_a & C_a B_a & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C_a A_a^{N_p-1} B_a & C_a A_a^{N_p-2} B_a & \dots & C_a A_a^{N_p-N_c} B_a \end{bmatrix}$$

The objective of the Predictive Control System is to bring the predicted Process Response as close as possible to the Set-Point signal $(r_s)_k$ representing a measure of predictive performance. The objective is realized once a control input trajectory $u_{1:K}$ is found that minimizes an error function between the Set-Point and the predicted Process Response. The following cost function is introduced:

$$J = (R_s - Y)^T (R_s - Y) + \Delta U R_u \Delta U \tag{6.13}$$

Where $R_s = \overbrace{\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}}^{1 \times N_p} (r_s)_k$ and $R_u = r_u I_{N_c \times N_c}$ and where r_u is used as a tuning parameter for the desired closed-loop performance. For now r_u is selected to be equal to 0.1. Meaning there are limitations on how large the input can be. To find an optimal ΔU, the Cost Function is minimized with respect to ΔU. The following Control Law is found:

$$\Delta U = (\Phi^T \Phi + R_u)^{-1} \phi^T (R_s - F(x_a)_k) \tag{6.14}$$

As previously mentioned the desired Set-Point is $r(k) = 15$ [m/s]. The running time of the simulation is selected to be 50 [s]. Which translates into $k = 500$ Sampling Instants, corresponding to the selected Sampling time of dt= 0.1 [s]. A realistic result is achieved when the Prediction Horizon is selected as $N_p = 100$ and the Control Horizon is selected as $N_c = 2$. For the selection of the Prediction Horizon, in case of a badly damped and unstable Process, the following lower bound applies $N_p = 2\text{int}(2\omega_s/\omega_b)$. Where ω_s is the Sampling frequency [49]. The Bandwidth for this particular problem is however very small such that the formula for the lower bound suggests a huge Prediction Horizon. A smaller Prediction Horizon than $N_p = 100$ results into a high Overshoot and therefore oscillations which does not correspond to how a car accelerates. The increasing of N_p often improves stability and performance of the closed-loop. A higher Prediction Horizon than $N_p = 100$ yields to a failing convergence towards the Set-Point. The increasing of the Control Horizon does not have any effect worth mentioning which is presumably due to the simplicity of the problem. In general, a small Control Horizon has a smoothing effect of the control input trajectory. Plus does a small Control Horizon establishes a quick convergence to the steady state value of the control input trajectory. The Control Horizon should be selected invariably equal or higher than the order of the Process and equal to or smaller than the Prediction Horizon divided by two: $n \leq N_c \leq N_p/2$. This concludes all theory needed to implement the MPC Design.

In figure 6.2 the result of the implementation of the MPC Design is displayed. The Control trajectory, the lower subplot, is generated based on the estimated Process Response. The upper subplot, displays the convergence of estimated Process Response towards the desired Set-Point Reference. This first implementation serves solely to generate a control input trajectory which can be applied to the True Process -the simulation- to gain a "True" Process Response. This Process Response is required for the process of Learning according to the principle of Free Energy. The MPC Design therefore still uses the state space model derived from first principles. The estimated Process Response is a product of this state space model as well.

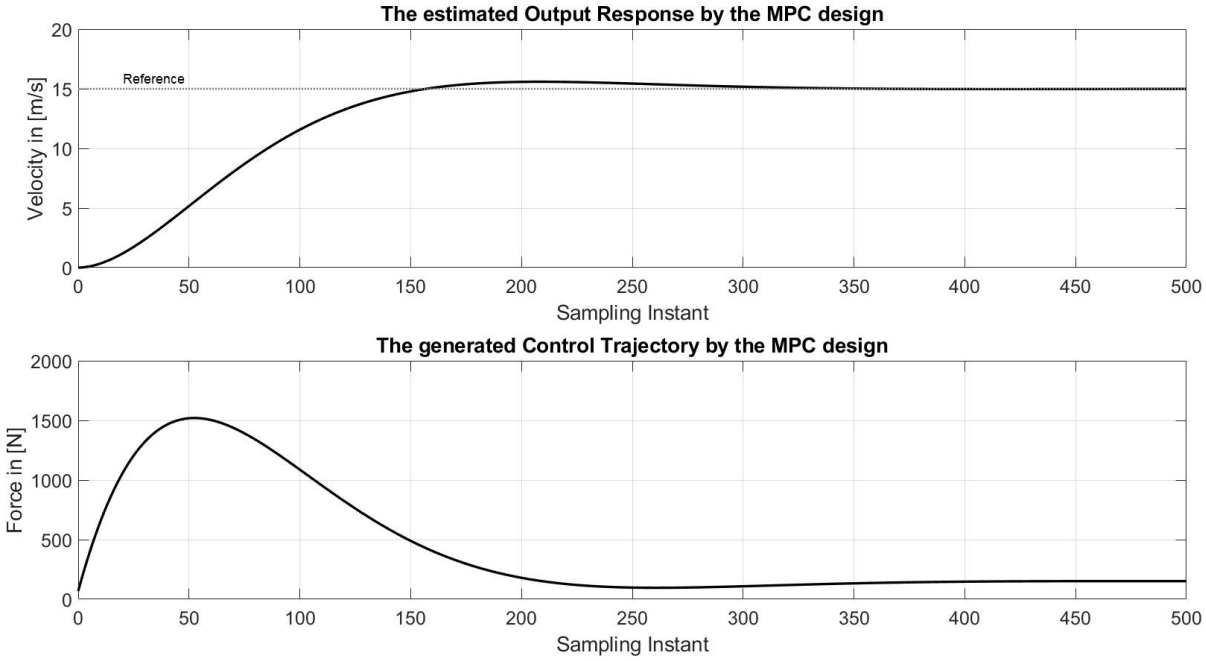


Figure 6.2: The generated control input trajectory and estimated output Response by the Model Predictive Control Design implementation

The generated control input trajectory is applied to the "True" state space representation of the Process, the discretized versions of equations 6.4 and 6.5 to gain a Process Response. The Process and Measurement Noises are stochastic processes. A stochastic process, in contrast to deterministic signals, is a family of random variables. These are all defined on the same Probability space and are therefore a statistical phenomenon. Each realization of the stochastic process is a unique function. The set of all possible realizations of the stochastic process is called an ensemble [7]. Each run of the MATLAB code yields a different realization of the noise sequences and are therefore a different realization of the Process Response. In figure 6.3 six different realizations of the Process Response are compared to the Noise-Free Response. The subplots show quite some variance with respect to each other, but also variance in the performance of the generated control input trajectory. In table 6.2 a quantitative measure of the deviation, the Mean Squared Error of the Process Response with respect to the Noise-Free Response for each realization is displayed. The Mean Squared Error values correspond to the deviations observed in figure 6.3. The main conclusion is that the model used in the MPC Design implementation is insufficient. The MPC Design implementation does not take the Process and Measurement Noise into account. This problem will be handled later on in this chapter. Furthermore, the realizations are used to prove the noise levels and performance of the MPC Design to have no effect on the performance of the Learning process.

Realization:	1	2	3	4	5	6
MSE in [%]	0.51	1.79	0.33	1.08	0.19	0.28

Table 6.2: The variance of the Process output Response realizations with respect to the Noise-Free Response

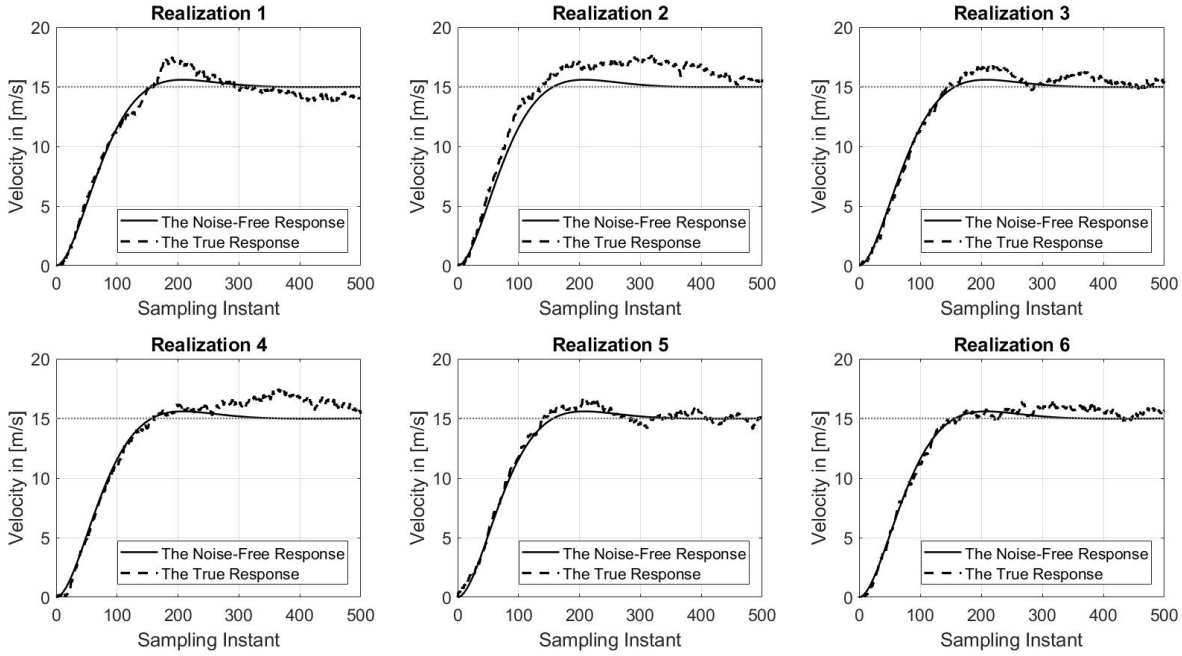


Figure 6.3: Various realizations of the True Process output Response to the generated control input trajectory compared to the Noise-Free Response

Now that various realizations of the Process output Response are generated it is time to implement the derived equations from chapter 4 and 5 for the state x and model matrices A and B.

6.3. THE PROCESS OF LEARNING ACCORDING TO THE PRINCIPLES OF FREE ENERGY

Remember the Free Energy equations with respect to A and B to be:

$$\begin{aligned}\mathscr{F}_A = &- \langle x_k\rangle^T \bar{\Sigma}_x^{-1} \mu_A \langle x_{k-1}\rangle - \langle x_{k-1}\rangle^T \mu_A^T \bar{\Sigma}_x^{-1} \langle x_k\rangle + \langle x_{k-1}\rangle^T \mathrm{tr}(\sum_{n=1}^{n} (\Sigma_A + \mathrm{vec}(\mu_A)\mathrm{vec}(\mu_A)^T)\{n,n\}\bar{\Sigma}_x^{-1})\langle x_{k-1}\rangle \\ &+ \langle x_{k-1}^T\rangle \mu_A^T \bar{\Sigma}_x^{-1} \langle B\rangle u_{k-1} + u_{k-1}^T \langle B\rangle^T \bar{\Sigma}_x^{-1} \mu_A \langle x_{k-1}\rangle - \ln(|\Sigma_A|) \end{aligned} \tag{6.15}$$

$$\begin{aligned}\mathscr{F}_B = &- \langle x_k\rangle^T \bar{\Sigma}_x^{-1} \mu_B u_{k-1} + \langle x_{k-1}\rangle^T \langle A\rangle^T \bar{\Sigma}_x^{-1} \mu_B u_{k-1} - u_{k-1}^T \mu_B^T \bar{\Sigma}_x^{-1} \langle x_k\rangle + u_{k-1}^T \mu_B^T \bar{\Sigma}_x^{-1} \langle A\rangle \langle x_{k-1}\rangle \\ &+ u_{k-1}^T \mathrm{tr}(\sum (\Sigma_B + \mathrm{vec}(\mu_B)\mathrm{vec}(\mu_B)^T)\{n,p\}\bar{\Sigma}_x^{-1}) u_{k-1} - \ln(|\Sigma_B|) \end{aligned} \tag{6.16}$$

In this formulation of both Free Energy equations, some time instances are neglected for the sake of clarity. The implementation of the Free Energy cannot be executed without these time instances and, therefore, are included back into the formulation.

Furthermore, in the case of one-dimensionality, the model matrices are scalars which simplify the calculations. The Free Energy equations become:

$$(\mathscr{F}_A)_k = \prod_{k=1}^{K} (-2(\bar{\Sigma}_x^{-1})_{k-1}\langle x_k\rangle\langle x_{k-1}\rangle(\mu_A)_k + (\bar{\Sigma}_x^{-1})_{k-1}\langle x_{k-1}\rangle^2((\Sigma_A + (\mu_A)^2)_k + 2(\bar{\Sigma}_x^{-1})_{k-1}\langle x_{k-1}^T\rangle\langle B\rangle u_{k-1}(\mu_A)_k - \ln(|(\Sigma_A)_k|)) \tag{6.17}$$

$$(\mathscr{F}_B)_k = \prod_{k=1}^{K} (-2(\bar{\Sigma}_x^{-1})_{k-1}\langle x_k\rangle u_{k-1}(\mu_B)_k + (\bar{\Sigma}_x^{-1})_{k-1}(u_{k-1})^2(\Sigma_B + (\mu_B)^2)_k + 2(\bar{\Sigma}_x^{-1})_{k-1}\langle x_{k-1}\rangle\langle A\rangle u_{k-1}(\mu_B)_k - \ln(|(\Sigma_B)_k|)) \tag{6.18}$$

To have some sense of what kind of measure the Free Energy is, both the Free Energy equations are calculated for each Sampling Instant using the variables and statistical moments that belong to the "True" Process. The model matrices $A_d = (\mu_A)_{1:K}$ and $B_d = (\mu_B)_{1:K}$ are assumed to be constant. The variance terms Σ_A and Σ_B are equivalent to zero. From the simulation that generated the "True" Process Response to the generated control input trajectory the "True" state is found. By a straightforward calculation, the corresponding variance $\bar{\Sigma}_x$ is obtained. Figure 6.4 displays the "True" Free Energy with respect to model matrices A and B:

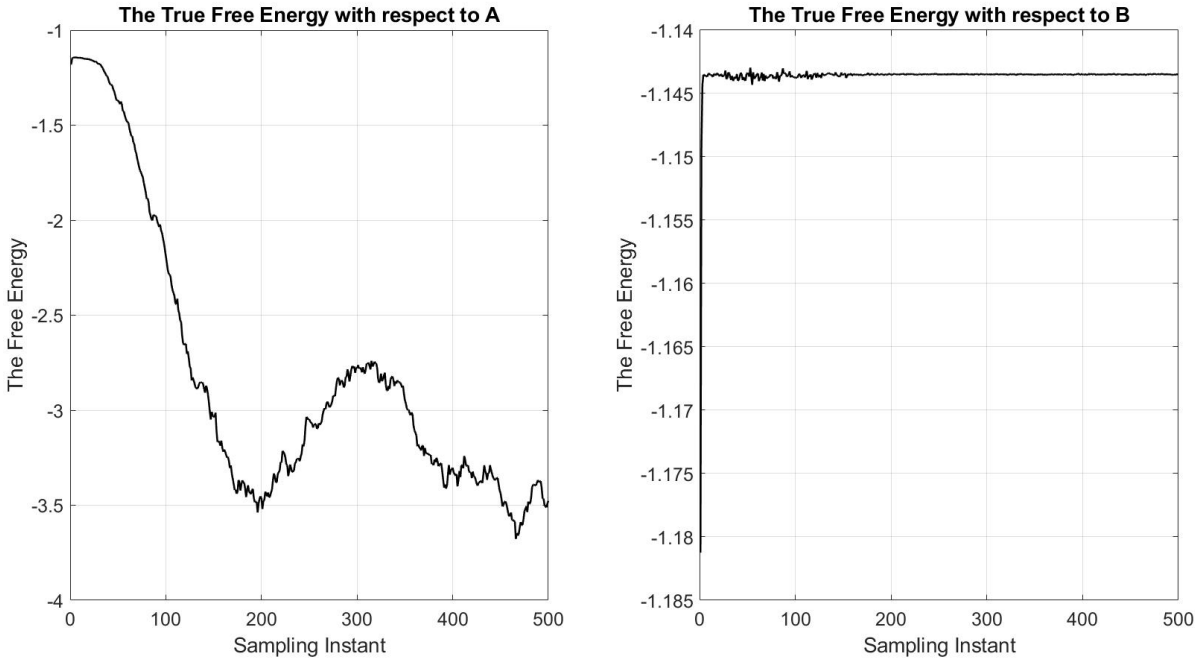


Figure 6.4: The True Free Energy associated to the True Process

Both the Free Energies show to become roughly constant after 200 Sampling Instants which is in correspondence with the course of the control input trajectory. A small test is executed with wrong values for the A matrix, respectively: $\begin{bmatrix}0.001 & 0.01 & 0.1 & A_d & 2 & 5\end{bmatrix}$. For the randomly selected Sampling Instant $k = 250$. It is checked whether the Free Energies corresponding to the wrong A matrix are indeed larger than the Free Energy corresponding to the correct A matrix.

A matrix:	0.001	0.01	0.1	A_d	2	5
The Free Energy with respect to A	-1.15	-1.19	-1.58	-3.44	-1.16	33.19
The Free Energy with respect to B	-1.145	-1.146	-1.146	-1.144	-1.141	-1.133

Table 6.3: The Free Energy value for Sampling Instant: $k = 250$ corresponding to the wrong and correct A matrix

Table 6.3 shows that indeed the the Free Energy with respect to A is the smallest for the correct A matrix and therefore enhance the confidence in the derived equations of chapter 4 and 5. Furthermore, the plots showed in figure 6.4 are used later on in this chapter as reference for the Free Energies which will be calculated with the estimated values.

To test whether the derived equations yield to good results, the required model matrices in both the process of state estimation: the Expectation step and the process of Learning the model matrices: the Maximization step are assumed to be the model matrices derived by first principles and provided. Which is a bit odd because this implies that in the Maximization step the process of Learning the model matrices the correct model matrices are already provided. But also, the process of state estimation is performed using the correct set of model matrices. This part therefore solely serves to check whether the derived equations yield to correct result in the presence of a more complete set of information. Remember the Free Energy Learning algorithm to evolve entirely around retrieving the full set of information: the state and model matrices in an iterative manner. Where a full set of information corresponds to the maximization of the Likelihood Probability and therefore minimization of Surprisal. When the correct model matrices are provided the Expectation step only needs one iteration to gain a "good" state estimate. The process of state estimation in fact estimates the noise levels present in the True process and uses these to correct a initial state estimation from the provided Noise-Free state space model representation. Multiple iterations are only required once the Noise-Free state space model is unknown which will be discussed later on in this chapter. Remember the process of state estimation according of the principles of Free Energy under given assumptions to be equivalent to the Kalman Filter algorithm. The graphs displayed in figure 6.5 are, therefore as well a result of an implementation of the Kalman filter algorithm.

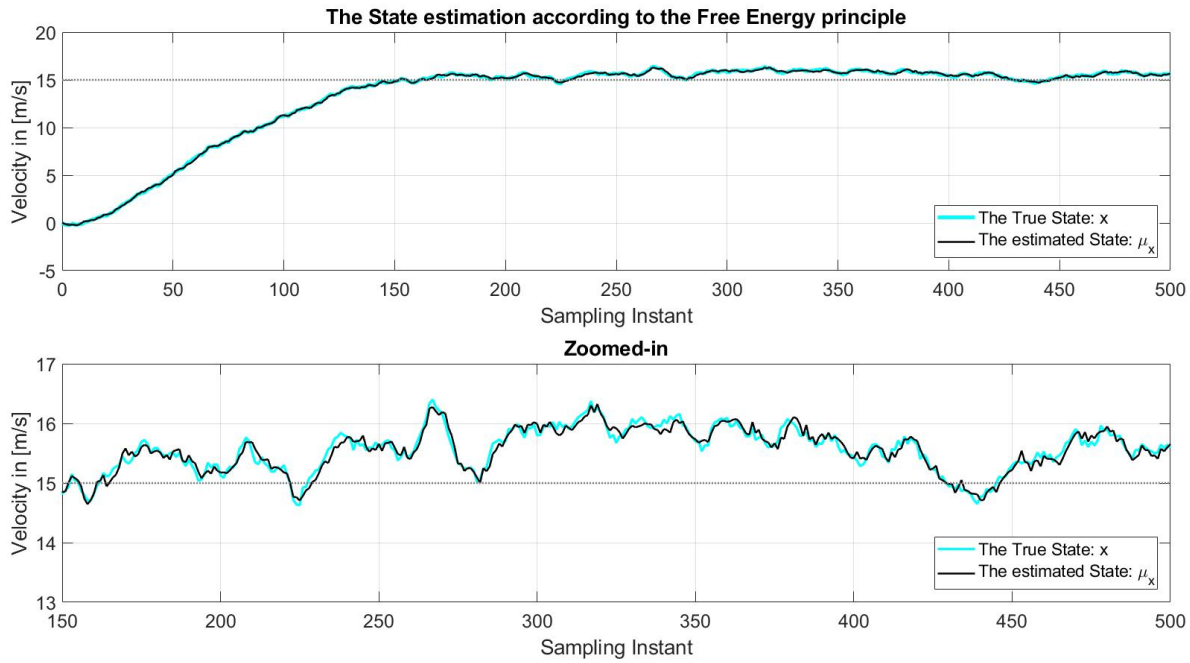


Figure 6.5: A comparison of the True Process and estimated state according to the principles of Free Energy

All of the developed MATLAB scripts are attached in appendix A. From figure 6.5 it can be concluded that the process of state estimation performs properly. The estimation of the state μ_x and corresponding variance Σ_x are used in the Maximization step: the Learning of the model matrices. Because of the "good" state estimation and provided correct model matrices the process of Learning will also yield to good results once the derivations in chapter 4 and 5 are done correctly.

Equations 6.19 and 6.20 are used in the process of Learning the model matrices: A and B:

$$(\Sigma_A)_k = \left(\prod_{k=1}^{K} (\bar{\Sigma}_x)_k^{-1} \left((\mu_x)_{k-1} (\mu_x)_{k-1} \right) \right)^{-1} \qquad (\mu_A)_k = (\Sigma_A)_k \prod_{k=1}^{K} (\bar{\Sigma}_x)_k^{-1} \left((\mu_x)_k (\mu_x)_{k-1} - \langle B \rangle u_{k-1} (\mu_x)_{k-1} \right) \tag{6.19}$$

$$(\Sigma_B)_k = \left(\prod_{k=1}^{K} (\bar{\Sigma}_x)_k^{-1} \left(u_{k-1} u_{k-1} \right) \right)^{-1} \qquad (\mu_B)_k = (\Sigma_B)_k \prod_{k=1}^{K} (\bar{\Sigma}_x)_k^{-1} \left((\mu_x)_k u_{k-1} - \langle A \rangle (\mu_x)_{k-1} u_{k-1} \right) \tag{6.20}$$

Where the variable u represents the generated control input trajectory. The vectors μ_A and μ_B contain the estimations of model matrices A and B: $\mu_A = \langle A \rangle$ or A_e and $\mu_B = \langle B \rangle$ or B_e for each Sampling Instants. The first few Sampling Instant suffer from convergence and therefore do not produce accurate model matrices. The convergence is showed in table 6.4. The correct model matrices are $A_d = 0.999$ and $B_d = 9.995 \cdot 10^{-5}$.

k=	0	1	2	3	4	5	6	7	8	9	…	499	500
$\langle A \rangle_k$	0	-2.52	0.97	1.37	1.06	1.11	1.17	1.19	1.09	1.04	…	0.999	0.999
$\langle B \rangle_k \cdot 10^{-3}$	0	-0.21	-0.07	-0.05	-0.01	-0.01	-0.01	-0.01	0	0.01	…	0.01	0.01

Table 6.4: The convergence of the model matrix estimates per Sampling Instant

The last estimations in the mean vectors: $(\mu_A)_{\text{end}}$ and $(\mu_B)_{\text{end}}$ in combination with the state estimation -the complete information set- are used to obtain a prediction of the Process output Response:

$$(\hat{y})_{0:K} = A_e(\mu_x)_{0:K} + B_e(u)_{0:K}$$

Figure 6.6 show a comparison between the "True" and estimated Process Response. From the figure 6.6 can be concluded that, also, the process of Learning the model matrices performs correctly.

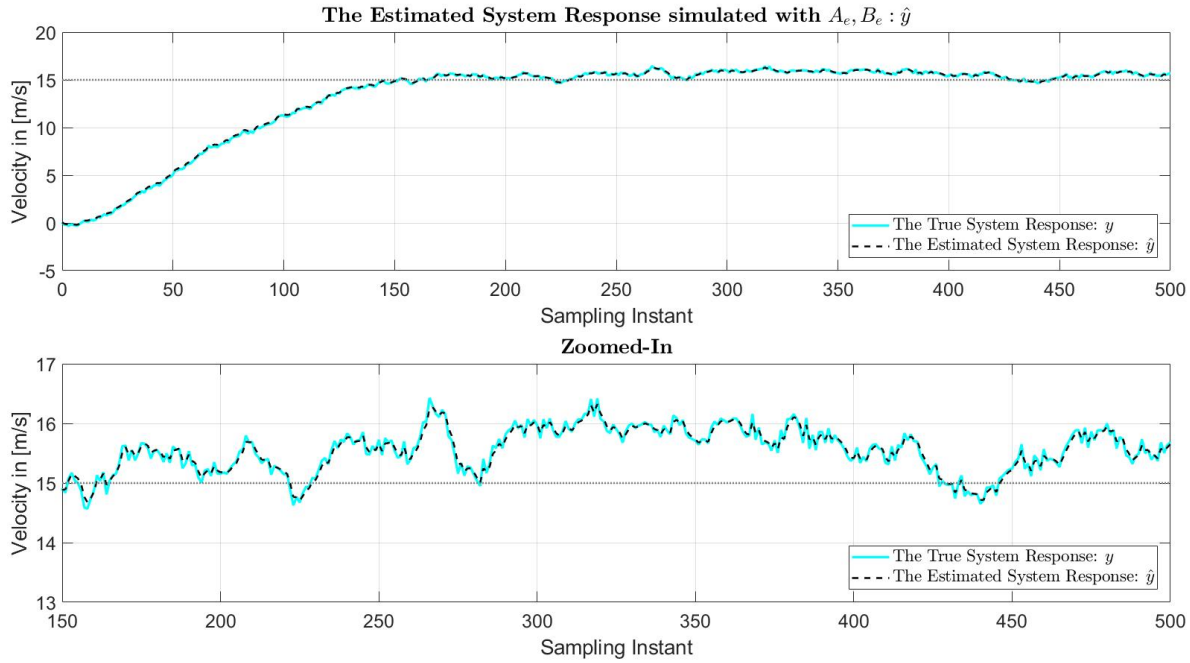


Figure 6.6: A comparison of the True Process Response with the estimated Process Response

The implementations of the Expectation step, the state estimation and Maximization step, the Learning of the model matrices is tested on the different realizations that were displayed in figure 6.4. In figure 6.7 and 6.8 the state estimation compared to the "True" Process state and the estimated Process Response compared to the "True" Process Response are showed for different realizations. From both figures 6.7 and 6.8 can be concluded that the different realizations do not have any effect on the performance of the Free Energy Learning algorithm. This makes sense in that the Free Energy Learning algorithm utilizes the information associated to each realization. It therefore purely estimates the specific Process Response and does not take into account yet the desired set-point. In the next part of this chapter, the Free Energy Learning algorithm is incorporated with the MPC Design, which will improve the performance of the MPC Design and therefore yield less variance in the different realizations as were displayed in figure 6.4. The figures 6.7 and 6.8 solely serve to prove that despite the variance in the performance of the MPC Design the performance of the Free Energy Learning algorithm remains constant and good.

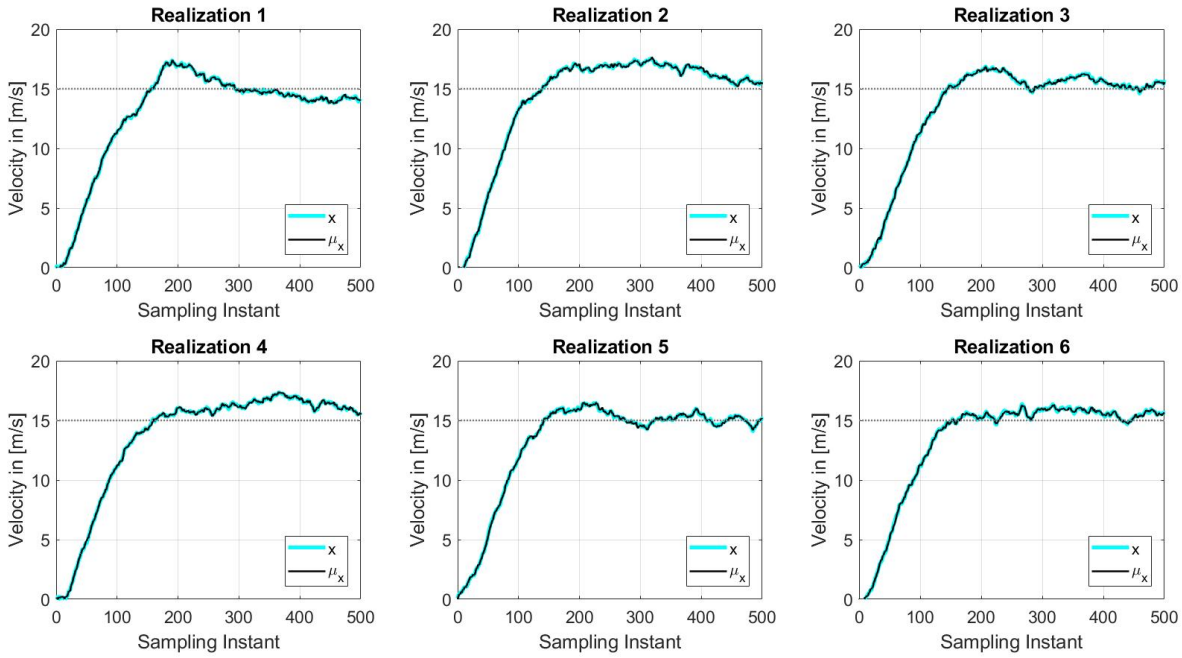


Figure 6.7: A comparison of the True Process and estimated state according to the principles of Free Energy for different realizations

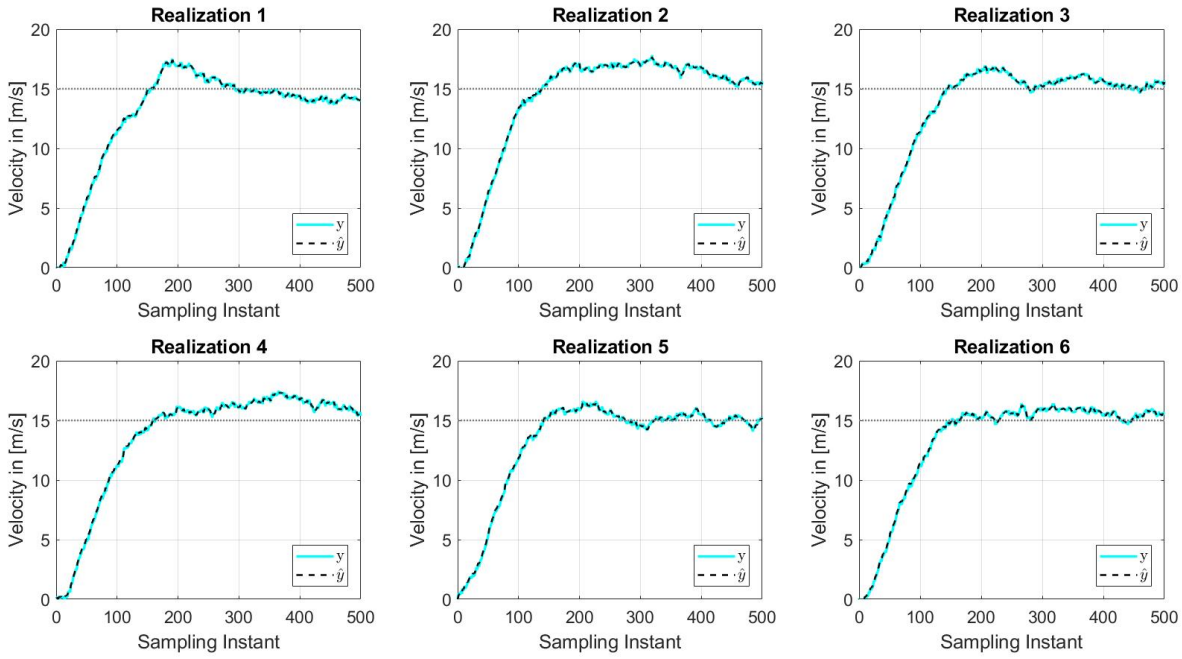


Figure 6.8: A comparison of the True Process Response with the estimated Process Response for different realizations

To provide a firm basis for the drawing of conclusions, the graphical results are accompanied by a quantitative model evaluation measure. The percentage **V**ariance **A**ccounted **F**or is an example of a measure of how well a model can describe an unknown system. Or in other words, a measure of how much of the variance in the output can be explained by the model [53]. The VAF is defined as:

$$\text{VAF} = \left(1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)}\right) 100\%$$

Depending on the MATLAB script run the VAF is equal to $\approx 99\%$, which means that the estimations are indeed accurate. In table 6.5 the VAF for each realization is displayed, which complies with previous statement that different realization do not have any effect on the performance of the Free Energy Learning algorithm.

Realization:	1	2	3	4	5	6
VAF in [%]	99.968	99.971	99.970	99.972	99.967	99.969

Table 6.5: A quantitative model validation by using the VAF measure for each different realization

While the result and conclusions are certainly satisfactory, the performance of the Free Energy Learning algorithm is excellent, it should not be forgotten that both the Expectation and Maximization steps are in possession of the correct model matrices.

The estimations of the means μ_A and μ_B and the variances Σ_A and Σ_B are the statistical moments of the recognition densities: $q(A)$ and $q(B)$. In the following figure 6.9 both the Recognition Densities are displayed. The "True" values of the model matrices are $A_d = 0.999$ and $B_d = 9.99510^{-5}$.

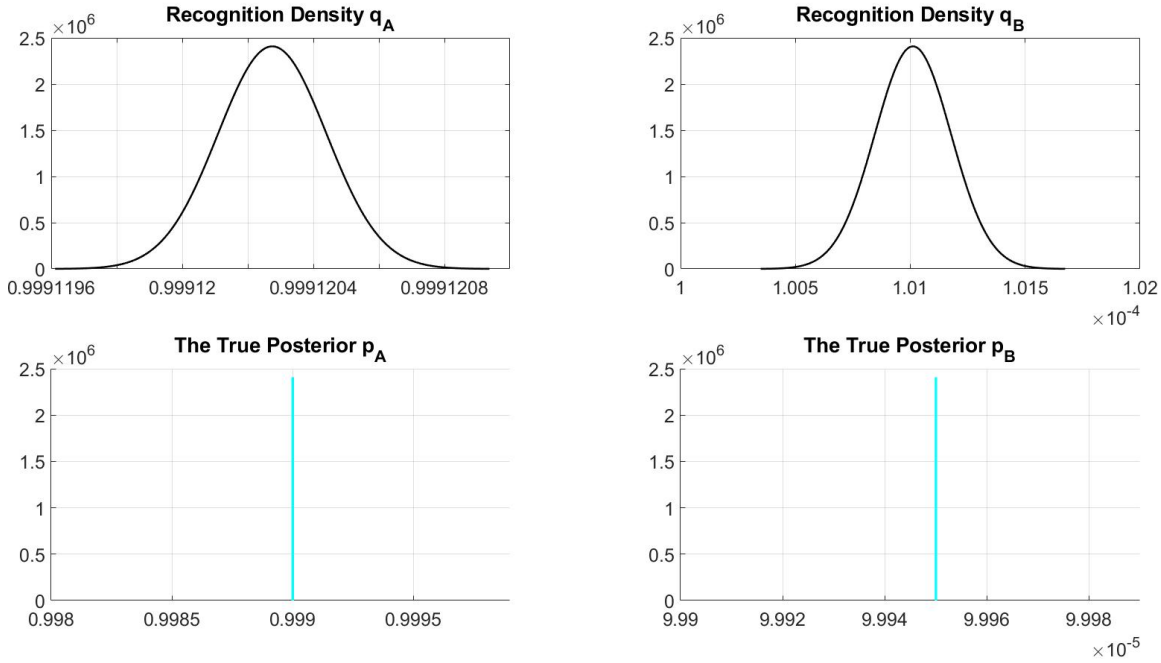


Figure 6.9: A visualization of the Recognition Density and True Posterior

All the plots displayed in figure 6.9 are Gaussian distributed. A Gaussian distribution or alternatively known as a Normal distribution is a function of the two statistical moments: the mean and the variance. Where the mean represents the place of the maximum of the Gaussian distribution and the variance represents a measure for the width of the Gaussian distribution. Remember from chapter 3 that the Free Energy Learning algorithm essentially attempts to reduce the divergence between an auxiliary distribution q_A or q_B and the True Posterior p_A and p_B. Where the auxiliary distribution is a Gaussian approximation of the True Posterior. In this case the "True" Posteriors p_A and p_B are assumed to be constant and, therefore, not to enclose any model uncertainty. The variance of the model matrices is equivalent to zero. In case of a Real set-up the visualization of the True Posteriors in figure 6.9 would be significantly different. The figure 6.9 would then be a lot more interesting. Figure 6.9 primarily shows that the variances of the Recognition Densities are indeed small where they should be equivalent to zero. The small offset in the variances is presumably due to small errors in the MATLAB calculations.

6.4. THE MPC DESIGN EMBEDDED WITH THE FREE ENERGY LEARNING ALGORITHM

The algorithm is proven to yield accurate results for both the process of state estimation as for the process of Learning the model matrices. The next step is to involve the Model Predictive Controller. Instead of the provided model matrices derived by first principle the estimated model matrices are incorporated into the MPC Design such that an iterative control-loop results. The full Model-Control algorithm is summarized in table 6.6 displayed on the next page.

The full Model-Control algorithm performs the control estimation of the state and Learning of the model matrices in the same sequence as is presented in the previous part of this chapter. Some randomly selected initial matrices are used to perform Model Predictive Control. In the previous implementation of the Model Predictive Controller a very simple version of state estimation was embedded to gain a representation of the Augmented state: x_a. This simple version of state estimation is replaced by the state estimation according to the Free Energy principle, the Kalman filter algorithm. The script `MPCFreeEnergyStateEst` used within the Model Predictive Control function *MPCControl* is a simplification of the actual script used in step 4 of the algorithm presented in table 6.6. Step 4 actually is redundant once the script `FreeEnergyStateEst` is used within the Model Predictive Controller. To clearly distinguish the different steps, step 4 is still present in the algorithm presented in table 6.6. A minor modification of the scripts is required once step 4 will be omitted from the full Model-Control algorithm. The Model Predictive Controller gains a Control input trajectory and the "True" Process Response which are used collectively with the initial model matrices to gain new estimates of these model matrices. With these estimations of the model matrices the model validation measure: VAF is calculated. The current estimates of the model matrices are used in the next iteration until the VAF measure is high enough and the While-loop will stop. The Learned model matrices are now proper estimations of the actual model matrices.

Algorithm 1:	1-D Model Predictive Control embedded with the Free Energy Learning algorithm at the end of iteration analogous with Batch Learning
Step 0:	**Set initial conditions:**
	Set Control and Prediction horizons: N_p and N_c
	Set Initial model matrices: $A_0 \rightarrow \mu_A$ and $B_0 \rightarrow \mu_B$
	Set Initial state and state variance: $(\mu_x)_0$ and $(\Sigma_x)_0$
	Set desired set-points: R_s and R_u
	While VAF $\leq$ 99% or # of Iterations $\leq$ Maximum # of Iterations
	Perform step 1,2,3,4,5&6
Step 1:	**Perform Model Predictive Control:**
	function $[u_{1:K}, y_{1:K}]$ =`MPCControl`$(\mu_A, \mu_B, (\mu_x)_0, (\Sigma_x)_0, R_s, R_u, N_p, N_c)$
	- Calculate Augmented state space: $[A_a, B_a]$ =`AugmentedSS`(μ_A, μ_B)
	- Calculate the F and Φ matrix: $[F, \Phi]$ =`MPCmatrices`(A_a, B_a, N_p, N_c)
	For: $k = 1$ till selected # of Sampling instants: K:
	- Simulate the "True" Process Response and "True" state:
	$[y_k, x_k]$ =`simRealResponse`(u_{k-1}, x_{k-1})
	- Save the "True" Process Response: $y_{1:K}(k) = y_k$
	- Calculate state estimation according to Free Energy Principle:
	$[(\mu_x)_k, (\Sigma_x)_k]$ = `MPCFreeEnergyStateEst`$(\mu_A, \mu_B, u_{k-1}, (\mu_x)_{k-1}, (\Sigma_x)_{k-1})$
	- Set the Augmented state: $(x_a)_k = \begin{bmatrix} (\mu_x)_k - (\mu_x)_{k-1} \\ y_{k-1} \end{bmatrix}$
	- Calculate the Control input trajectory according to the Control law:
	$\Delta U = (\Phi^T \Phi + R_u)^{-1} \phi^T (R_s - F(x_a)_k)$ then $u_k = \Delta U(1,1) + u_{k-1}$
	End
	End
Step 2:	**Save:** $\mu_A, \mu_B, u_{1:K}$ in the data file: Sys
Step 3:	**Perform state estimation according to the Free Energy Principle:**
	i.e. The Kalman Filter algorithm
	$[(\mu_x)_{1:K}, (\Sigma_x)_{1:K}, (\bar{\Sigma}_x)_{1:K}]$ = `FreeEnergyStateEst`(Sys, $y_{1:K}, (\mu_x)_0, (\Sigma_x)_0$)
	$(\mu_x)_0 = (\mu_x)_{1:K}$(begin) and $(\Sigma_x)_0 = (\Sigma_x)_{1:K}$(begin)
Step 4:	**Calculate the Free Energies:** $\mathscr{F}_a$ and $\mathscr{F}_B$
Step 5:	**Perform model matrices Learning according to the Free Energy Principle:**
	$[(\mu_A)_{1:K}, (\Sigma_A)_{1:K}, (\mu_B)_{1:K}, (\Sigma_B)_{1:K}] = \ldots$
	`FreeEnergymodelParameterEst1D`(sys, $y_{1:K}, (\mu_x)_{1:K}, (\bar{\Sigma}_x)_{1:K}$)
	$\mu_A = (\mu_A)_{1:K}$(end) and $\mu_B = (\mu_B)_{1:K}$(end)
Step 6:	**Calculate the VAF**
	End

Table 6.6: Algorithm 1: One-Dimensional MPC Design embedded with the Free Energy Learning algorithm at the end of iteration analogous with Batch Learning

The initial model matrices are randomly selected as $A_0 = [0.2]$ and $B_0 = [0.1]$. The rest of the initial conditions remain the same value as was already specified. In this particular run the algorithm needs eight iterations to obtain a VAF percentage higher than 99%. In figure 6.10, the "True" Process Response for each iteration which has a VAF percentage between 0% and 100% is plotted. An iteration is alternatively known as an epoch which is terminology used in Machine Learning and represents a Learning cycle. Meaning each Sampling Instant in the set has been seen.

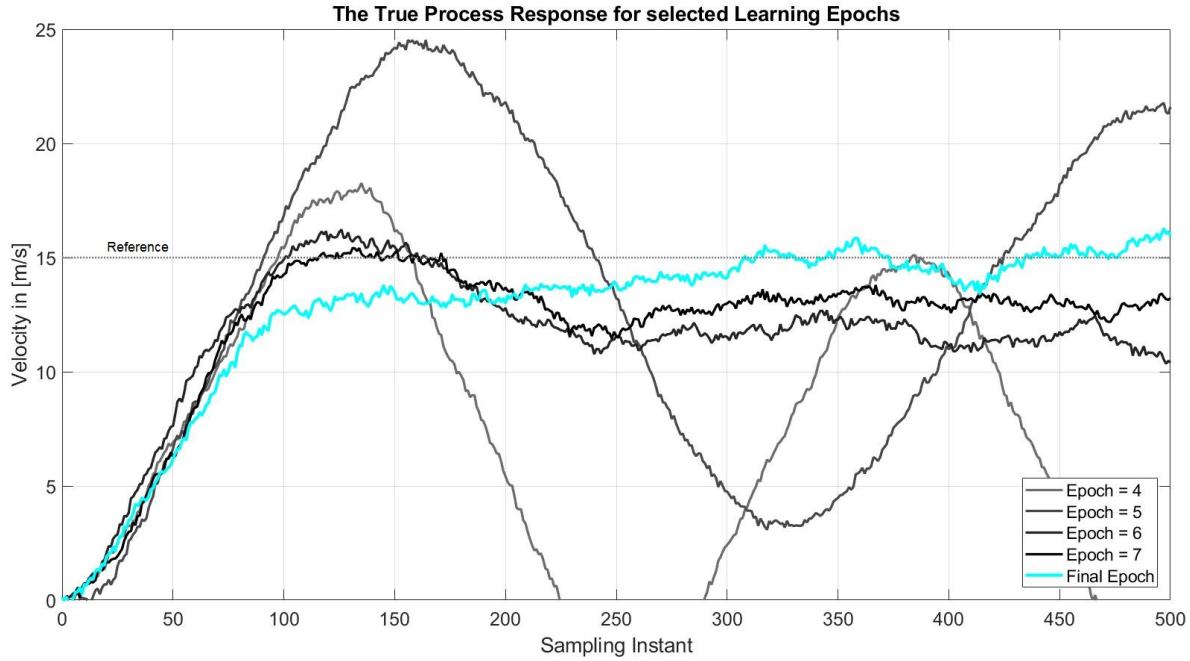


Figure 6.10: The True Process Response convergence of the control-loop, the True Process Response for VAF-selected iterations

The figure 6.10 shows a convergence in the capability of the Control input trajectory associated to each epoch to bring the Process Response to the reference Set-point. The Control input trajectory is generated with the estimated model matrices. The Control input trajectory associated with the last Epoch has the "best" estimation of the model matrices. The same conclusion can be drawn from the numerical results displayed in table 6.7. While the model error of the A matrix is only a small percentage of the True model matrix: $|A_d - A_e| = 0.0019$ the model error of the B matrix is a relatively large percentage of the True model matrix: $|B_d - B_e| = 7.28 \cdot 10^{-5}$. This however does seem to not have a major effect on the estimation of Process Response as can be concluded from the VAF percentage. If the while-loop is allowed more iterations the estimate of the B matrix becomes better. These iterations are not required to obtain a satisfying estimation of the Process Response.

Epoch	1	2	3	4	5	6	7	8
A_e	-0.758	-0.373	-0.0349	0.665	0.817	0.978	0.986	0.997
B_e	0.045	0.0074	0.0031	0.0023	0.0007	0.0007	$1.9 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$
VAF in [%]	-211.96	-93.39	-84.84	17.23	69.55	94.54	98.04	99.72

Table 6.7: The convergence of the control-loop expressed in numbers

In step 6 of the algorithm presented in table 6.6 the Free Energies with respect to the model matrices A and B are calculated. In figure 6.11 the Free Energies for each Epoch which has a VAF percentage between 0% and 100%, is plotted compared to the "True" Free Energy as plotted in figure 6.5.

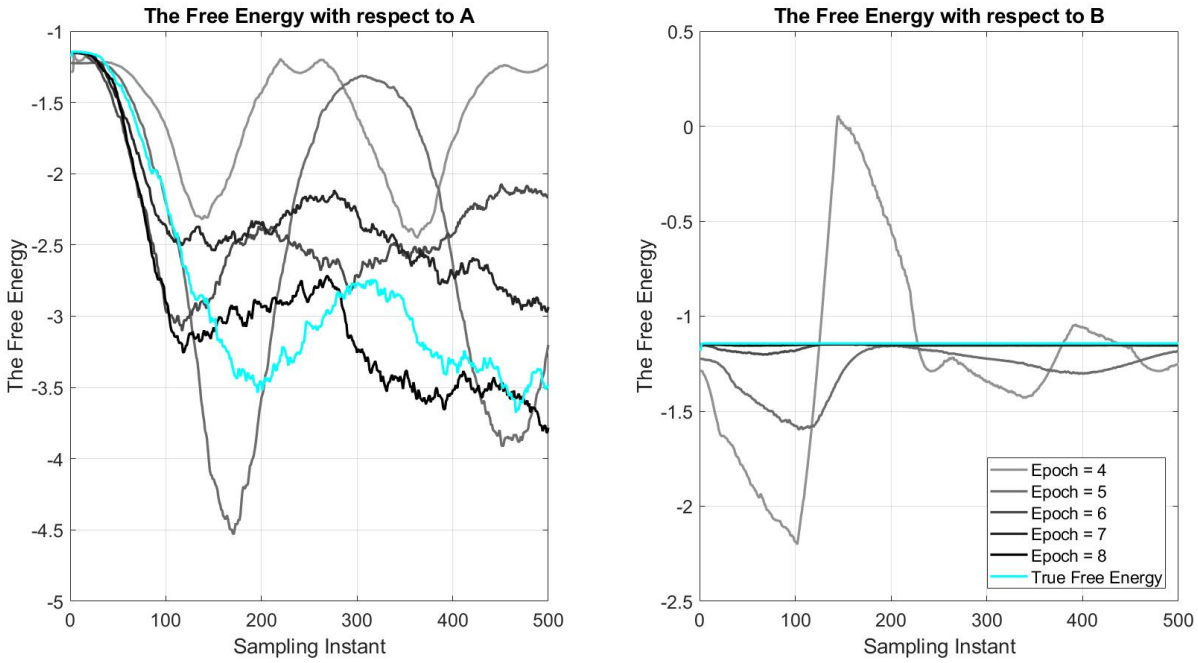


Figure 6.11: The Free Energy convergence of the control-loop, the Free Energy with respect to model matrix A and B for VAF-selected iterations

Both Free Energies calculations for several Epochs converge to the "True" Free Energies. The better the model estimates of model matrices are the better the estimates of the Free Energy are. Whether the Free Energy is actually minimized is difficult to conclude from the representation in the figure: 6.11. It requires more experience with the Free Energy construct to draw a conclusion.

To ensure that various realizations yield to similar result and the "True" Process Response converges to the Set-point, the MATlAB script is run twelve times. In figure 6.12 the VAF percentage convergence is plotted for each realization, which shows a almost identical coarse. In figure 6.13 the "True" Process Response is plotted, which shows that for each realization the Control input Trajectory brings the "True" Process Response towards the desired Set-point. The model matrix estimations and VAF percentages like displayed in table 6.7 are close to identical for each realization. Each realization follows a close to identical coarse of convergence. The difference in the "True" Process Response are related to different noise levels associated with each realization and the MPC Design capability of dealing with these noises. While the performance is somewhat better in relation to the realizations displayed in figure 6.3 the improvement is minor. The improvement was expected to be more significant because the noises are taking into account. A more advanced MPC Design presumably would yield to a better performance such that the True Process Response remains in the Set-Point. Because the model matrix estimates are close to identical for each realization the deviation in the True Process Response cannot be accounted to the performance of the Free Energy Learning algorithm. The robustness of the Free Energy Learning algorithm is high.

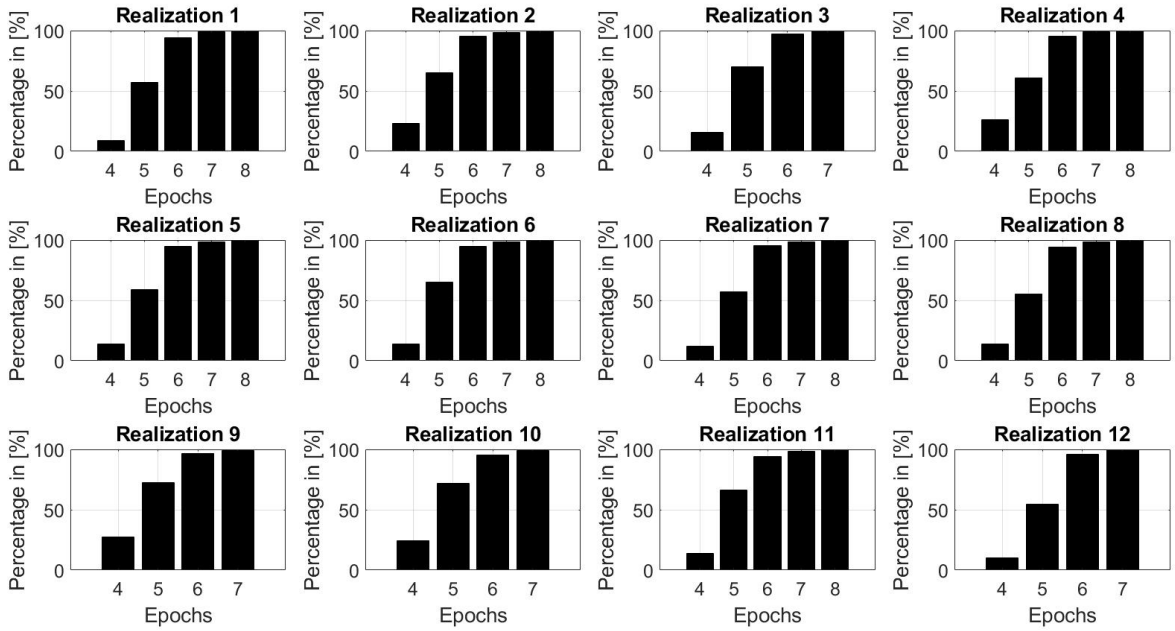


Figure 6.12: The Variance Accounted for percentage convergence for different realizations

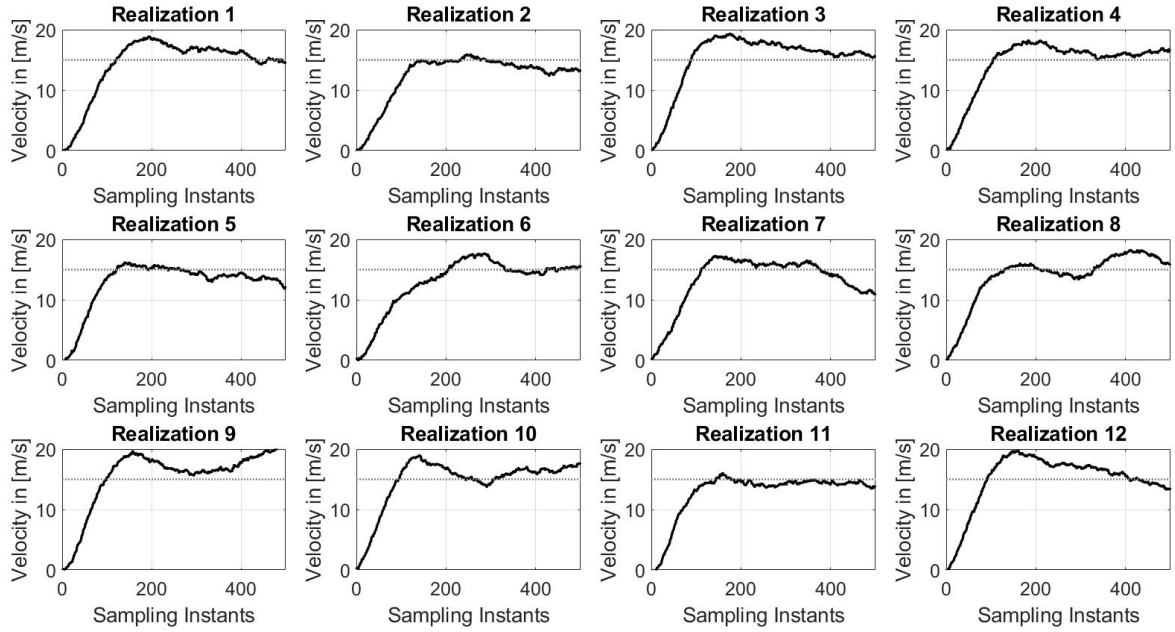


Figure 6.13: The True Process Response of the Model-Control loop generated control input for different realizations

6.4.1. THE SIMULTANEOUS LEARNING OF THE MODEL MATRICES WITHIN THE MPC DESIGN

Up till now the Learning of the model matrices has been performed at the end of each Learning Epoch. To start with, a set of data is generated and at once delivered to the MATLAB script Learning the model matrices. This process corresponds to Batch Learning and is proven to yield proper results. One can however imagine that the whole process of several Epoch to be slow and therefore to not always be suitable in case of Real-Time Applications. The process of model matrix Learning, fortunately, is also suitable to perform in a simultaneous manner. The process of control, state estimation and Learning of the model matrices is performed simultaneously. The process of model matrix Learning only requires the current and one step-back state and state variance, the current model matrix estimates and one step-back control input. This makes it suitable to incorporate into the MPC Design. The algorithm displayed in table 6.6 is modified into the following table 7.5.

Algorithm 2:	1-D Model Predictive Control embedded with the Free Energy Learning algorithm in an simultaneous manner
Step 0:	**Set initial conditions:**
	Set Control and Prediction horizons: N_p and N_c
	Set Initial model matrices: $A_0 \rightarrow \mu_A$ and $B_0 \rightarrow \mu_B$
	Set Initial state and state variance: $(\mu_x)_0$ and $(\Sigma_x)_0$
	Set desired set-points: R_s and R_u
	Create Laguerre matrix: L, with scaling factor $0 < a < 1$ and N_p, N_c
Step 1:	**Perform Model Predictive Control:**
	function $[u_{1:K}, y_{1:K}, \hat{y}_{1:K}, \text{Sys}] = \ldots$
	`MPCControlFullLaguerre`$(\mu_A, \mu_B, (\mu_x)_0, (\Sigma_x)_0, R_s, R_u, N_p, N_c, L)$
	For: $k = 1$ till selected # of Sampling instants: K:
	- Simulate the "True" Process Response and "True" state:
	$[y_k, x_k] =$`simRealResponse`(u_{k-1}, x_{k-1})
	- Save the "True" Process Response: $y_{1:K}(k) = y_k$
	- Calculate state estimation according to Free Energy Principle:
	$[(\mu_x)_k, (\Sigma_x)_k] =$ `MPCFreeEnergyStateEst`$(\mu_A, \mu_B, (\mu_x)_{k-1}, (\Sigma_x)_{k-1}, y_k, u_{k-1})$
	- Save the estimated Process Response: $\hat{y}_{1:K}(k) = (\mu_x)_k$
	- Calculate model matrix estimations:
	$[(\mu_A)_k (\mu_B)_k] = \ldots$
	`FreeEnergyModelParametersEstMPC`$((\mu_A)_{k-1}, (\mu_B)_{k-1}, (\mu_x)_{k-1}, (\mu_x)_k, u_{k-1})$
	- Save the model matrix estimates: Sys.A= $\lvert(\mu_A)_k\rvert$ and Sys.B= $\lvert(\mu_B)_k\rvert$
	- Calculate Augmented state space: $[A_a, B_a] =$`AugmentedSS`(Sys)
	- Calculate the Ω_k and Ψ_k matrix
	- Set the Augmented state: $(x_a)_k = \begin{bmatrix} (\mu_x)_k - (\mu_x)_{k-1} \\ y_k - R_s \end{bmatrix}$
	- Calculate the control input trajectory according to the Control Law:
	$\Delta U = -L^T(\Omega_k^{-1}\Psi_k)(x_a)_k$ then $u_k = \Delta U(1,1) + u_{k-1}$
	- Save the control input: $u_{1:K}(k) = u_k$
	End
	End

Table 6.8: Algorithm 2: One-Dimensional MPC Design embedded with the Free Energy Learning algorithm in an simultaneous manner

The primary advantage of the algorithm proposed in table 7.5 is that unforeseen changes in the environment or Process Dynamics, the disturbances and model uncertainties are directly accounted for. After evaluation of the algorithm with the basic implementation of the MPC Design it is discovered that the performance of the MPC Design is not sufficient enough for the algorithm to work properly. The second Algorithm is more dependent on the performance of the MPC Design than the first algorithm. The involvement of Laguerre functions is known to, significantly, improve the performance of the MPC Design. In the next chapter 7 the Laquerre functions will be explained in more detail. For the present they are just implemented into the second algorithm. The "True" Process Response for various realizations is displayed in figure 6.14.

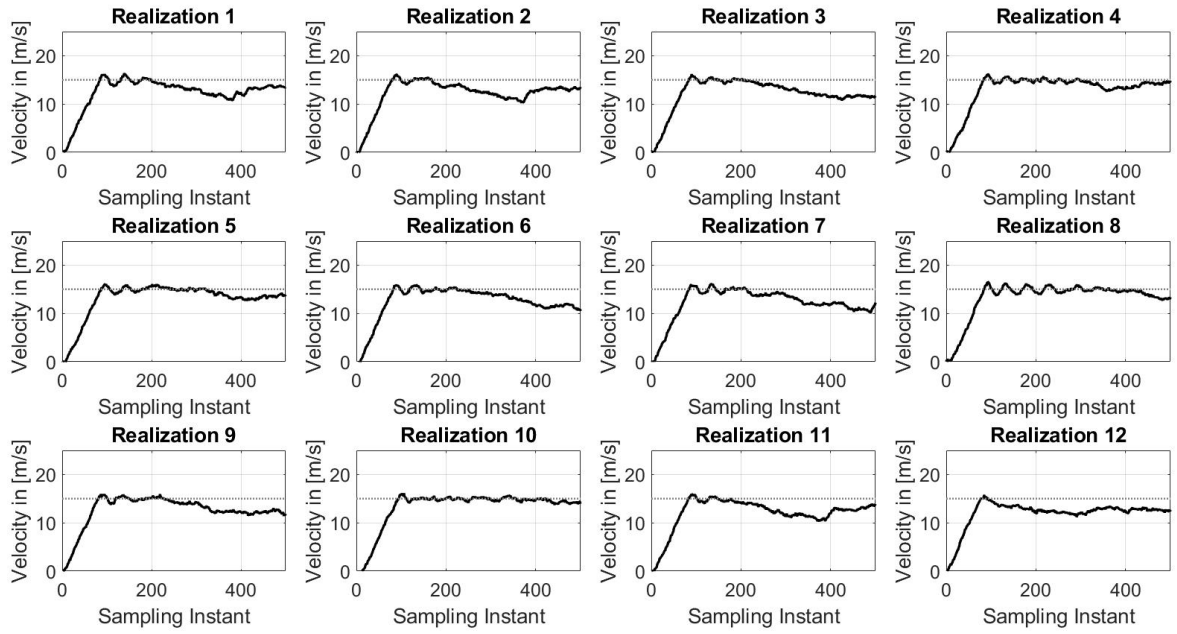


Figure 6.14: The True Process Response for various realization acquired by the second algorithm which performs control, state estimation and model matrix Learning simultaneously

Once more, the "True" Process Response is brought to the desired Set-point while in each plot there is some jitter present. In the following table 6.9 the estimated model matrices associated to each realization are displayed.

Realization	1	2	3	4	5	6	7	8	9	10	11	12
A_e	0.96	0.97	0.97	0.96	0.97	0.96	0.96	0.92	0.96	0.92	0.96	0.99
$B_e \cdot 10^{-3}$	0.5	0.4	0.5	0.6	0.5	0.5	0.5	1	0.5	1	0.4	0.1

Table 6.9: The convergence of the control-loop expressed in numbers

From figure 6.14 the "True" Process Response realization 10 could be seen as the best performance of the full Model-Control algorithm. The model matrix estimations associated with realization 12 are however the best estimations. The performance of the MPC Design is therefore once again believed to have a big share in the performance of the full Model-Control algorithm. A study into improvement of the MPC Design will expectantly yield a better performance. Furthermore, the model matrix estimations become more accurate once more Sampling Instants are considered. This is however at the expense of a larger computational time.

6.5. DISCUSSION AND CONCLUSION

The Free Energy Learning algorithm is a very versatile and intuitive algorithm suitable for either Batch Learning after a control input trajectory is generated and applied or for the simultaneous performing of control, state estimation and model matrix Learning. The message of the Free Energy Learning algorithm is clear. Once a full set of information is obtained, the creation of predictions becomes easy. This is indeed showed in the first implementation of the Free Energy algorithm where the estimations where fantastic. It reveals the power of the algorithm once the current information: the control input Trajectory, the True Process Response, the state estimations and the model matrix estimations become better and better in case of a more realistic case where the initial information provided is not yet accurate. Both Model-Control algorithms presented haven proven to yield accurate results. There is some jitter present in the realization of the "True" Process Response, displayed in both the figures 6.13 associated to the first Model-Control algorithm and 6.14 associated to the second Model-Control algorithm. This is believed to be a result of the selected basic MPC Design implementation, supported by the associated VAF percentages. A study into more advanced MPC Design methods could significantly improve the capability of the MPC Design to bring the "True" Process Response to the desired Set-point. It can also improve the computational burden. The matrices involved in the MPC Design are large. In addition, the Free Energy Learning algorithm and control input trajectory are in both Model-Control algorithms highly dependent, suggesting the increase of performance, presumably, positively effects the state estimation and model matrix Learning. The MPC Design is, however, not the focus point of this thesis. Besides that eventually the MPC Design is substituted by a control input trajectory generator according to the principles of Free Energy. This substitution should be rather easy once the control input trajectory generator according to the principles of Free Energy is specified.

The application of the implementations on a real set-up will, presumably, highlight the further possible improvements of both of the Model-Control algorithms. Only then one can validate whether the Free Energy Learning algorithm indeed can deal with model uncertainties and real-time adaptations.

The final chapter of this thesis, the Final Conclusion and Discussion will address the effect of the outlined assumptions in chapter 3. But first, the implementations will be extended to a more complex two-dimensional problem: the Pendulum on a Cart problem.

7
A TWO-DIMENSIONAL PROBLEM

7.1. THE PROBLEM: A PENDULUM-ON-A-CART

As a second illustration of the process of Learning according to the principles of Free Energy, the following "complex" two-dimensional problem is considered: a Pendulum-on-a-Cart. The Pendulum-on-a-Cart problem is a familiar control bench problem, alternatively known as the Inverted-Pendulum problem.

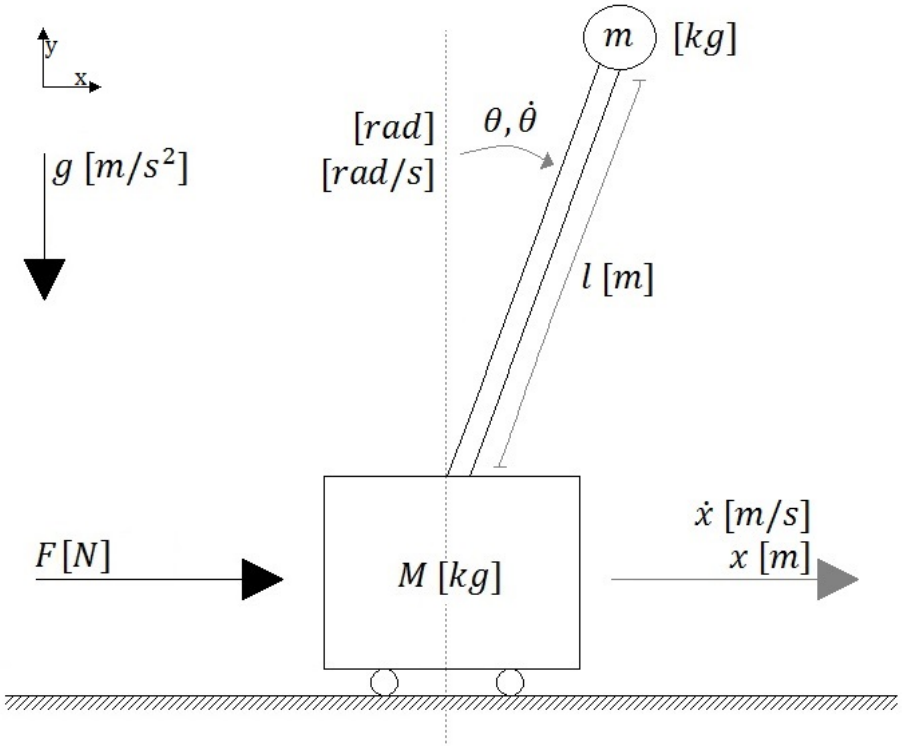


Figure 7.1: The true Process; A Pendulum-on-a-Cart problem

In the Inverted-Pendulum problem, the cart is usually actuated by a DC-motor: the input. The cart can move along one degree of freedom: the x-axis in figure 7.1. The Pendulum can rotate freely around the axis perpendicular to the x-y plane. The two measured outputs, which are to be controlled, are the position of the Cart and the angle of the Pendulum. The Process is **S**ingle **I**nput and **M**ultiple **O**utput.

The parameters of the Pendulum-on-a-Cart problem are summarized in the following table: 7.1,

Parameter	Value	Unit	Description
M	0.49	[kg]	Mass of the Cart
m	0.085	[kg]	Mass of the Pendulum
l	0.3	[m]	Length of the Pendulum
g	9.81	[m/s^2]	Gravity
C	10.42	[Ns/m]	Viscous Damping
I_m	0.3678	[kg m^2]	Motor Inertia
K_v	8.389	[Nm/A]	Gain conversion

Table 7.1: The parameters associated with the Pendulum-on-a-Cart problem

The parameters are retrieved from a former project carried out as part of the System & Control curriculum [10]. The values C, I_m and K_v represent the result of a simple System Identification implementation using the built-in MATLAB function: `greyest` with as input Process Response data generated on an actual Inverted Pendulum set-up.

A Linearized state space representation, derived from first principles, is used to simulate the Process Response data required for the state estimation and Learning of the model matrices according to the principles of Free Energy. The Continuous-Time model matrices are represented as [33]:

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{C}{I_m+M+\frac{m}{4}} & -\frac{3gm}{4I_m+4M+m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{3C}{4lI_m+4lM+lm} & \frac{\frac{3(g+3gm)}{4I_m+4M+m}}{4l} & 0 \end{bmatrix} \qquad B_c = \begin{bmatrix} 0 \\ \frac{K_v}{I_m+M+\frac{m}{4}} \\ 0 \\ -\frac{3K_v}{4l(I_m+M+\frac{m}{4}} \end{bmatrix} \tag{7.1}$$

$$C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad D_c = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{7.2}$$

The state vector is represented as:

$$x = \begin{bmatrix} \text{Position of the Cart:} & x \text{ in [m]} \\ \text{Velocity of the Cart:} & \dot{x} \text{ in [m/s]} \\ \text{Angle of the Pendulum:} & \theta \text{ in [rad]} \\ \text{Angular velocity of the Pendulum:} & \dot{\theta} \text{ in [rad/s]} \end{bmatrix}$$

The number of state variables contained in the state corresponds to the assumed order of the Process: $n = 4$. With MATLAB function: `c2d`, the Continuous-Time model is converted into a Discrete-Time model. The closed-loop Bandwidth frequencies of both control input to Cart and control input to Pendulum are assumed to be about 1 [rad/s]. Customarily, the Sampling Time is selected thirty times larger than the closed-loop Bandwidth. The Sampling Time, therefore, is selected as 0.01 [s] [33]. The simulation will be running for 10$[s]$ which translates into $K = 1000$ samples.

To achieve a "more" realistic simulation of the "True" Process, the state space equations are perturbed by a random Process and Measurement Noise. The variance of the Measurement Noise is selected as: $\Sigma_w = 0.01$ and the variance of the Process Noise is selected as: $\Sigma_z = 0.01$.

$$x_{k+1} = A_d x_k + B_d u_k + w_k \qquad w_k \backsim \mathcal{N}(0, \Sigma_w) \tag{7.3}$$

$$y_k = C_d x_k + z_k \qquad z_k \backsim \mathcal{N}(0, \Sigma_z) \tag{7.4}$$

7.2. THE MODEL PREDICTIVE CONTROL DESIGN

Any arbitrary input trajectory, like for example a Step-input, applied on the Process yields an unstable (Step-)Process Response. The Process Response grows to infinity. To generate a stable Process Response the same MPC Design as in chapter 6 is implemented.

In the case of multi-dimensional problems, Controllability of the process becomes a matter of concern; while this was not so for the problem discussed in chapter 6. A Process is said to be controllable if and only if, it is possible, by means of the control input, to transfer the system from any initial state to any other state in a Finite Time. As algebraic condition this means that the rank of Controllability matrix: $\begin{bmatrix} A & AB & \dots & A^{n-1}B \end{bmatrix}^T$ has to be equivalent to the rank of the Process [11]. The Controllability matrix should, therefore, have a rank equivalent to 4 which is the case. In chapter 6 only one dimension is considered. The Controllability matrix is equivalent to A and, therefore, cannot lose rank.

The desired angle of the Pendulum is set at zero, which corresponds to the unstable upright position. The desired position of the cart is set at 10 [m]; this position was chosen arbitrarily. The Prediction Horizon and Control Horizon are tuned and found to yield the best result when N_p is selected as 180 and N_c is selected as 10. This corresponds to the tuning rules presented in chapter 6 where the formula: $N_p = 2\text{int}(2\omega_s/\omega_b)$ counts as lower bound for the Prediction Horizon. The sample Frequency is equivalent to: $\omega_s = 1/\text{dt} = 100$ [rad/s] and with the assumed closed-loop Bandwidth to be about 1 [rad/s], let's say $\omega_b \approx 2$ [rad/s] the Prediction Horizon should indeed be about ≈ 200. The order of the Pendulum-on-a-cart problem is equivalent to $n = 4$. The Control horizon should be any number higher than $n = 4$ and lower than $N_p/2 \approx 100$.

In figure 7.2 the result of the implementation of the MPC Design is displayed.

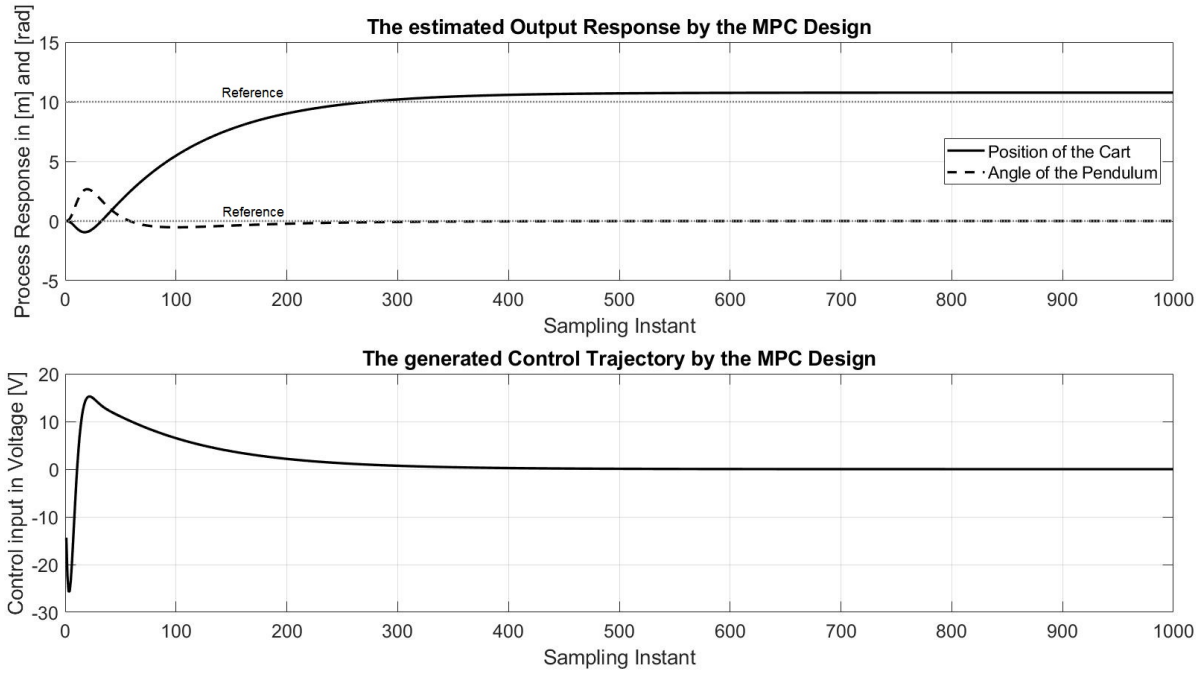


Figure 7.2: The generated control input trajectory and estimated output Response by the Model Predictive Control Design implementation

The figure 7.2 shows that the estimated position of the Cart does not exactly match the desired Set-Point of 10 [m]. The MPC Design fails to bring the Process Response to the desired Set-Points. A solution is to convert to a more advanced form of MPC Design. The currently implemented MPC Design is extended and reformulated to facilitate the use of Laguerre Functions. Laguerre functions are a set of Discrete orthonormal basis functions which can be used to structure the control input [49]. Reference [52] provides an excellent explanation of how the Laguerre functions are incorporated into the MPC Design. Figure 7.3 shows that MPC Design enhanced with Laguerre functions indeed yield to a more satisfactory performance. The Process Response reaches both the desired Set-Points. The generated control trajectory by MPC Design with Laquerre Functions maintains a similar form to the control trajectory by Classical MPC Design. There is only a minor difference in the two generated control Trajectories. It is, however, this minor difference that gets rid of the former present offset in the desired Set-Point of the Cart.

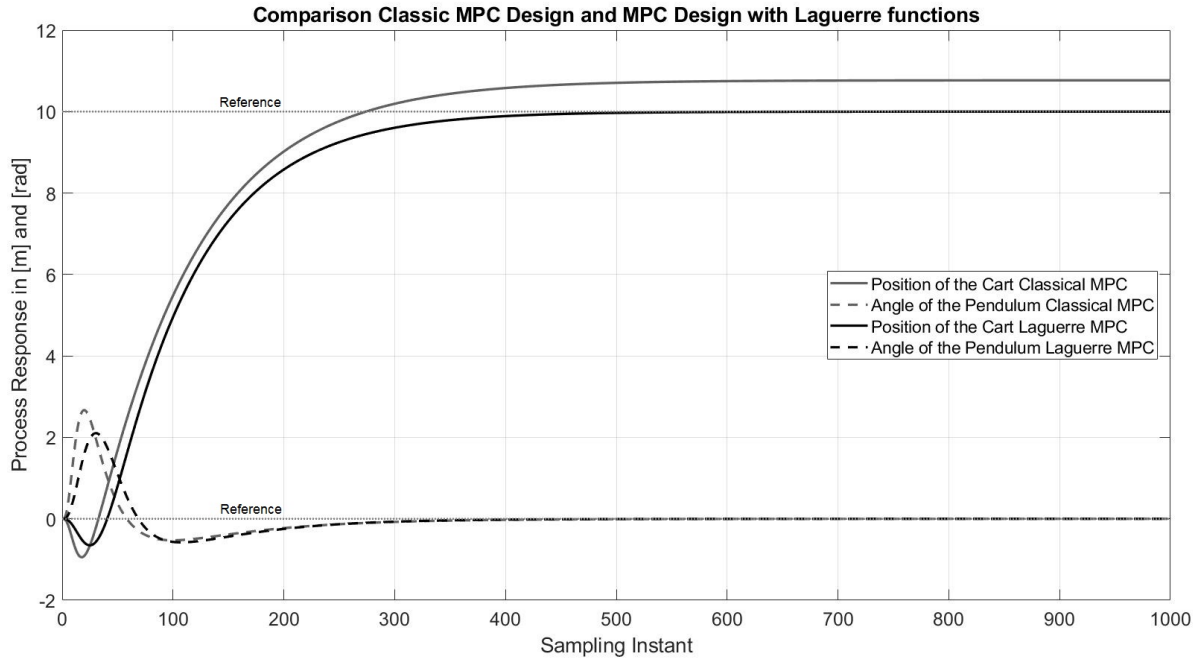


Figure 7.3: A comparison of the Performance of Classical and Model Predictive Control using Laguerre functions

To gain a "True" Process Response the generated control trajectory by Model Predictive Control using Laguerre functions is applied to the "True" Process. The entire True Process Response for every Sampling Instant is displayed in the upper plot of figure 7.4. It can be concluded that the generated control trajectory fails to bring the Process Response to the desired Set-Points. The Process Response grows to infinity. In the lower plot of figure 7.4 is shown that only for a few Sampling Instants the control trajectory yields good results. The "True" Process Response follows the by the MPC design estimated Noise-Free Process Response. The current MPC design does not, yet, take into account the presence of the noises. While at the first few Sampling Instants, the control trajectory is large enough to dominate the course of the Process Response, at higher Sampling instants, the control trajectory converges to zero, and the noises will dominate the course of the Process Response.

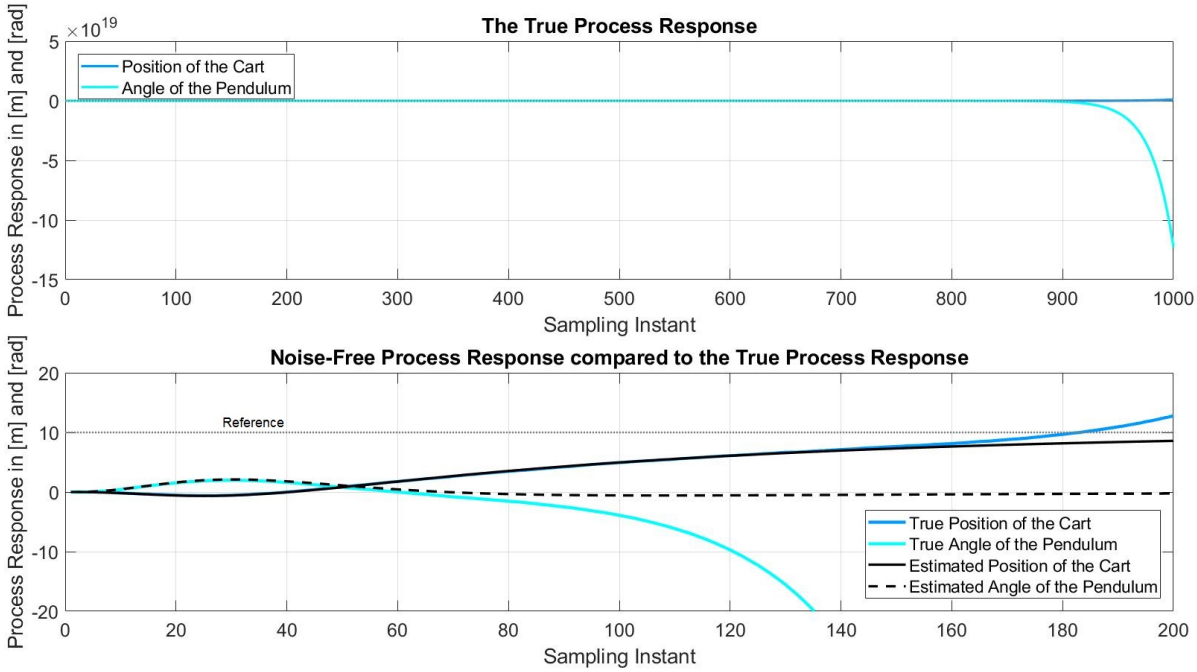


Figure 7.4: The True Process Response where the by MPC design using Laquerre Functions fails to bring the Process Response to the desired Set-Points

In chapter 6 it was established that the state estimation required for the Control Law could be performed according to the principles of Free Energy. The MPC Design incorporating Free Energy state estimation compensates for the present noises and, therefore, will yield a better performance. The following control trajectory is generated:

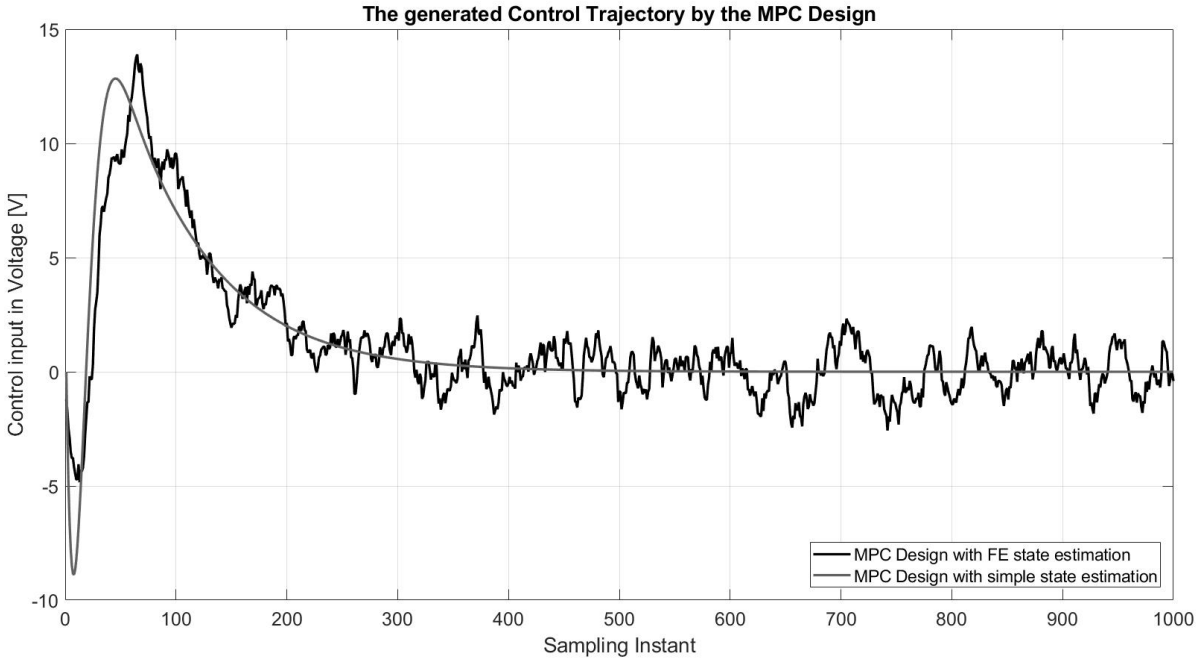


Figure 7.5: A comparison of the by Model Predictive Control generated control trajectory with and without noise compensation

The control trajectory thus generated is applied to the "True" Process. The "True" Process Response is displayed in figure 7.6. The control trajectory is capable of bringing and keeping the Process Response to the desired Set-Points. This is the Process Response data used in the Free Energy Learning algorithm.

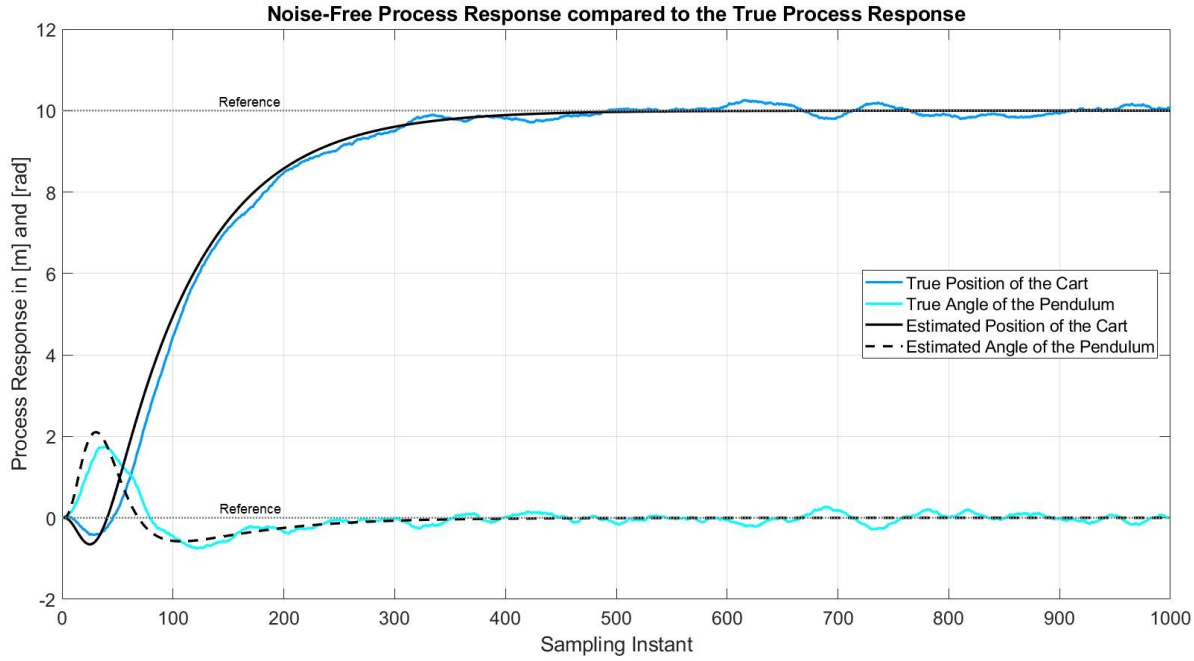


Figure 7.6: The True Process Response data used in the Free Energy Learning algorithm

7.3. THE PROCESS OF LEARNING ACCORDING TO THE PRINCIPLES OF FREE ENERGY

The Free Energy equations with respect to model matrices: A and B, where the neglected time instants are included back into the equations and where the one-dimensionality of the input is acknowledged, are:

$$
\begin{aligned}
(\mathscr{F}_A)_k = \prod_{k=1}^{K} \big(& - \langle x_k \rangle^T \bar{\Sigma}_x^{-1} (\mu_A)_{k-1} \langle x_{k-1} \rangle - \langle x_{k-1} \rangle^T (\mu_A)_{k-1}^T \bar{\Sigma}_x^{-1} \langle x_k \rangle \\
& + \langle x_{k-1} \rangle^T \mathrm{tr}(\sum ((\Sigma_A)_{k-1} + \mathrm{vec}((\mu_A)_{k-1}) \mathrm{vec}((\mu_A)_{k-1})^T) \{n, n\} \bar{\Sigma}_x^{-1}) \langle x_{k-1} \rangle \\
& + \langle x_{k-1}^T \rangle (\mu_A)_{k-1}^T \bar{\Sigma}_x^{-1} \langle B_{k-1} \rangle u_{k-1} + u_{k-1}^T \langle B_{k-1} \rangle^T \bar{\Sigma}_x^{-1} (\mu_A)_{k-1} \langle x_{k-1} \rangle - \ln(|(\Sigma_A)_{k-1}|) \big)
\end{aligned} \tag{7.5}
$$

$$
\begin{aligned}
(\mathscr{F}_B)_k = \prod_{k=1}^{K} \big(& - \langle x_k \rangle^T \bar{\Sigma}_x^{-1} (\mu_B)_{k-1} u_{k-1} - (\mu_B)_{k-1}^T \bar{\Sigma}_x^{-1} \langle x_k \rangle u_{k-1} + \mathrm{tr}(((\Sigma_B)_{k-1} + (\mu_B)_{k-1} (\mu_B)_{k-1}^T) \bar{\Sigma}_x^{-1}) u_{k-1}^2 \\
& + \langle x_{k-1} \rangle^T \langle A_{k-1} \rangle^T \bar{\Sigma}_x^{-1} (\mu_B)_{k-1} u_{k-1} + (\mu_B)_{k-1}^T \bar{\Sigma}_x^{-1} \langle A_{k-1} \rangle \langle x_{k-1} \rangle u_{k-1} - \ln(|(\Sigma_B)_{k-1}|) \big)
\end{aligned} \tag{7.6}
$$

Before proceeding to the actual implementation of equations 7.5 and 7.6, the Free Energy equations will be calculated for every sampling instant using the variables and statistical moments belonging to the "True" process. This will give some sense of what kind of measure the Free Energy is for this particular problem.

The model matrices: $A_d = (\mu_A)_{1:K}$ and $B = (\mu_B)_{1:K}$ are assumed to be constant while the variance terms: Σ_A and Σ_B are equivalent to zero. The "True" state is a known variable: it was calculated while simulating the "True" process response to the generated control input trajectory. By a straightforward calculation, the corresponding variance: $\tilde{\Sigma}_x$ is obtained. The figure 7.7, displays the "True" Free Energy with respect to model matrices A and B:

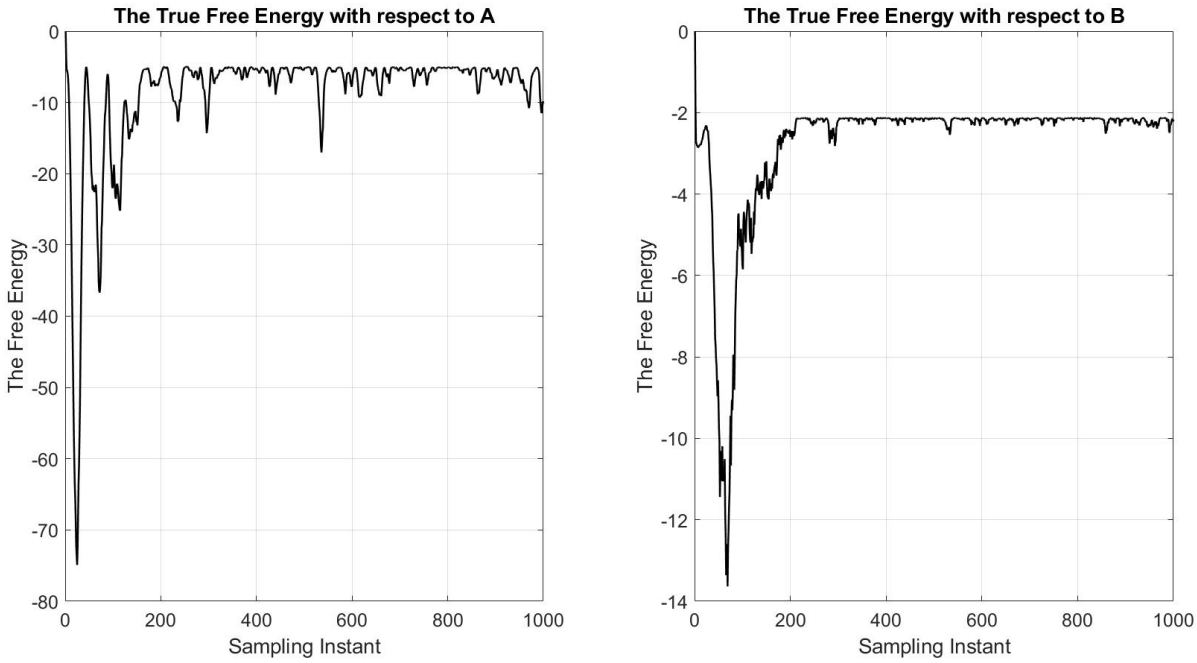


Figure 7.7: The True Free Energy associated to the True Process

Once again, the Free Energies show to become roughly constant after 200 Sampling Instants which is in correspondence with the course of the control input trajectory. The same small test as in chapter 6 is executed with a random A matrix. A random A matrix is obtained with random numbers on the entries. The random A matrix is weighted by different factors, respectively: $\begin{bmatrix}0.001 & 0.01 & 0.1 & 0.5\end{bmatrix}$. For the randomly selected Sampling Instant: $k = 250$, is checked whether the Free Energies corresponding to the wrong A matrix are indeed larger than the Free Energy corresponding to the correct A matrix.

Utilized A matrix:	$0.001 \cdot A_{\text{random}}$	$0.01 \cdot A_{\text{random}}$	$0.1 \cdot A_{\text{random}}$	$0.5 \cdot A_{\text{random}}$	A_d
The Free Energy with respect to A	-4.2628	-4.2629	-3.9420	4.5666	-5.1211
The Free Energy with respect to B	-2.0613	-2.0882	-2.3568	-3.5507	-2.2028

Table 7.2: The Free Energy value for Sampling Instant: $k = 250$ corresponding to the wrong and correct A matrix

The table 7.2 shows that indeed the Free Energy with respect to A is the smallest for the correct A matrix just as was proven in chapter 6. The Free Energy with respect to B should remain similar for any A matrix, albeit correct or incorrect. The B matrix remains correct. While the differences are subtle in comparison to the differences in the Free Energies with respect to A, they are more significant than the differences in Free Energies with respect to B from chapter 6 displayed in the table 6.3. The ratio of the random A matrix entry values with respect to the correct A matrix entry values is larger.

Once again the derived equations derived in chapters 4 and 5 are tested. Now however instead of dealing with scalars, matrices are involved increasing the complexity of the implementation. As in chapter 6, the required model matrices for both the Expectation step (state estimation) as well as the Maximization step (learning), are assumed to be arrived from first principes and provided. The values of the entries of the Discrete-Time model matrices: A_d and B_d are:

$$A_d = \begin{bmatrix} 1.00 & 0.0094 & 0.00 & 0.00 \\ 0.00 & -0.882 & -0.0067 & 0.00 \\ 0.00 & 0.0014 & 1.0013 & 0.01 \\ 0.00 & 0.2796 & 0.2621 & 1.0013 \end{bmatrix} \qquad B_d = \begin{bmatrix} 0.0005 \\ 0.09 \\ -0.0011 \\ -0.2251 \end{bmatrix} \tag{7.7}$$

Similar to chapter 6 the expectation step, only, needs one iteration to gain a "good" state estimation. Keep in mind the implementation of state estimation according to the Free Energy principle in this case to be an implementation of the Kalman filter. The result of the Expectation step is displayed in figure 7.8. As before, to chapter 6 the estimation is exceptionally accurate. This is, however, due to the extensive set of information made available. The state estimation algorithm is given the correct model matrices and noise variances: Σ_w and Σ_z. Later in this chapter, the Free Energy Learning algorithm will be discussed when less information is made available in corresponding to more realistic applications.

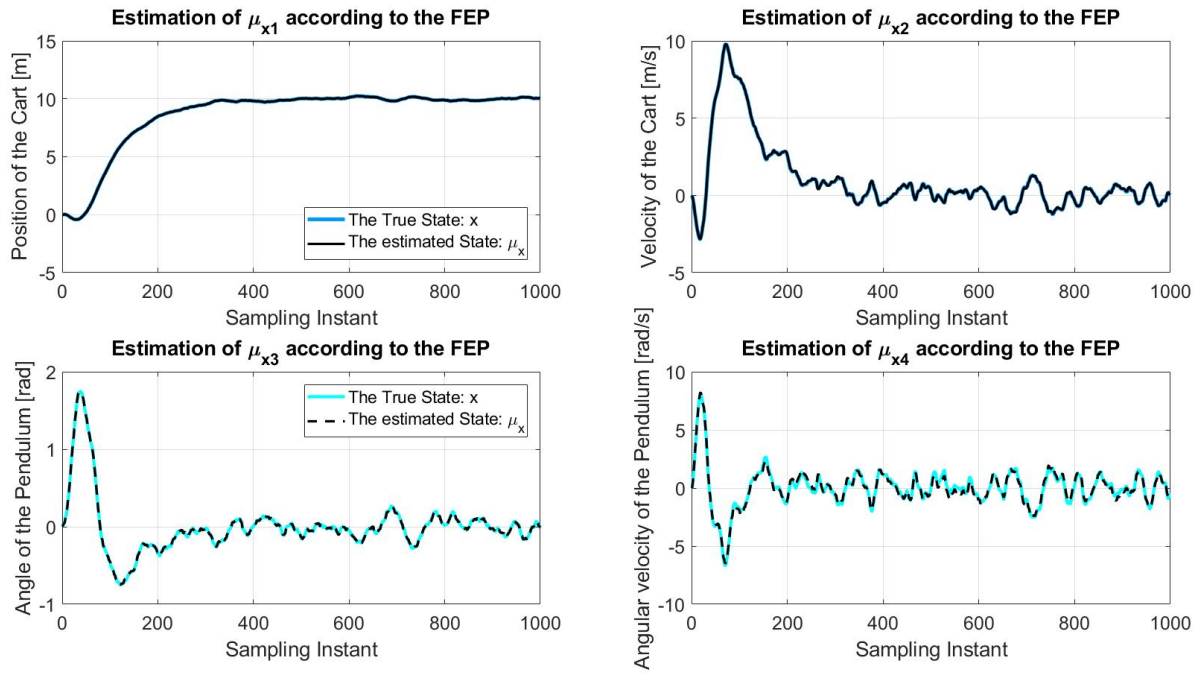


Figure 7.8: A comparison of the True Process and estimated state according to the principles of Free Energy

The estimation of the state: μ_x and corresponding variance: Σ_x are used in the Maximization step: the Learning of the model matrices. Because of the "good" state estimation and provided correct model matrices the process of Learning will also need only one iteration and presumably yield accurate results.

The following updated equations 7.8, 7.9, 7.10 and 7.11 are used in the process of learning the model matrices: A and B:

$$(\Sigma_A)_k = \left(\begin{bmatrix} (\bar{\Sigma}_x^{-1})_{k-1} & 0 & \dots & 0 \\ 0 & (\bar{\Sigma}_x^{-1})_{k-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & (\bar{\Sigma}_x^{-1})_{k-1} \end{bmatrix}^T \prod_{k=1}^{K} \langle x_{k-1} \rangle \langle x_{k-1} \rangle^T \right)^{-1} \tag{7.8}$$

$$(\mu_A)_k = \frac{\prod_{k=1}^{K} \langle x_k \rangle \langle x_{k-1} \rangle^T + \prod_{k=1}^{K} \langle B \rangle u_{k-1} \langle x_{k-1} \rangle^T}{\prod_{k=1}^{K} \langle x_{k-1} \rangle^T \langle x_{k-1} \rangle} \tag{7.9}$$

$$(\Sigma_B)_k = \left(\begin{bmatrix} (\bar{\Sigma}_x^{-1})_{k-1} & 0 & \dots & 0 \\ 0 & (\bar{\Sigma}_x^{-1})_{k-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & (\bar{\Sigma}_x^{-1})_{k-1} \end{bmatrix}^T \prod_{k=1}^{K} \langle u_{k-1} \rangle \langle u_{k-1} \rangle^T \right)^{-1} \tag{7.10}$$

$$(\mu_B)_k = \frac{\prod_{k=1}^{K} \langle x_k \rangle \langle u_{k-1} \rangle^T + \prod_{k=1}^{K} \langle A \rangle x_{k-1} \langle u_{k-1} \rangle^T}{\prod_{k=1}^{K} \langle u_{k-1} \rangle^T \langle u_{k-1} \rangle} \tag{7.11}$$

Where the variable u represents the generated control input trajectory. The vectors μ_A and μ_B contain the estimates of model matrices A and B: $\mu_A = \langle A \rangle$ or A_e and $\mu_B = \langle B \rangle$ or B_e, for each Sampling Instant. The first few Sampling Instants suffer from convergence and therefore do not produce accurate model matrices. With every $(\mu_A)_{1:K}$ and $(\mu_B)_{1:K}$ an estimation of the Process Response is made:

for $k = 1 : K$

$$(\hat{y})_{1:K} = (\mu_A)_k (\mu_x)_{0:K} + (\mu_B)_k (u)_{0:K}$$

end

In figure 7.9 the gained Process Response estimations are plotted excluding the first ten estimated Process Responses, which where chaotic and far from accurate. Figure 7.9 explains a convergence towards the actual "True" Process Response. All the plots show a similar course. This indicates that the model matrix entry estimates are dependent and capture the Process Dynamics even for the first few Sampling Instants but lack the correct gain. For the Inverted-Pendulum application, this means thereto, primarily, be weight on the diagonal of the A matrix. For the first few Sampling Instants, the A matrix estimation has the correct weight balance between the diagonal and the other entries. The value of the diagonal entries is, however, not yet correct. The last estimates in the mean vectors: $(\mu_A)_{\text{end}}$ and $(\mu_B)_{\text{end}}$ are selected as the final estimates. The following estimation of the model matrices is obtained:

$$(\mu_A)_{\text{end}} = \begin{bmatrix} 1.00 & 0.0095 & 0.0014 & 0.0001 \\ 0.00 & -0.882 & -0.0065 & 0.00 \\ 0.00 & 0.0018 & 1.0032 & 0.01 \\ -0.0001 & 0.2804 & 0.2666 & 1.0014 \end{bmatrix} \qquad (\mu_B)_{\text{end}} = \begin{bmatrix} 0.0006 \\ 0.09 \\ -0.009 \\ -0.2244 \end{bmatrix} \tag{7.12}$$

The difference with the "True" Process model matrices is $\approx 10^{-4}$ which can be considered as negligible.

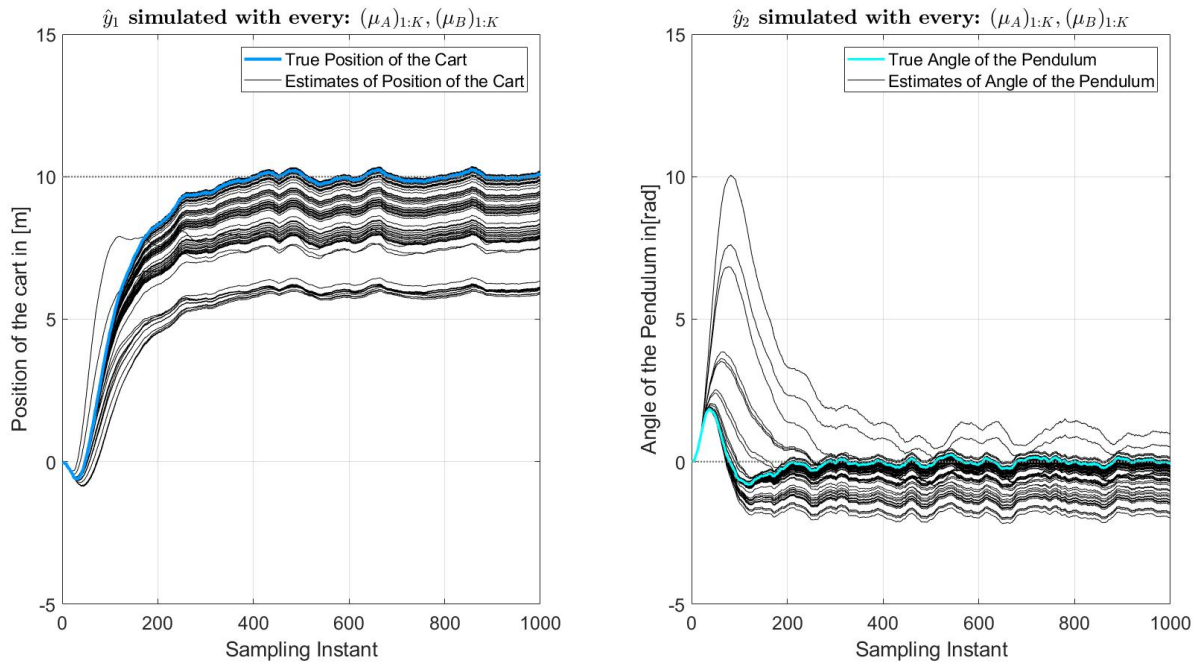


Figure 7.9: The convergence of the estimated Process Response simulated with every estimate of the model matrices compared to the True Process Response

In chapter 6, the Variance Accounted for was defined as:

$$\text{VAF} = \left(1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)}\right) 100\%$$

In table 7.3 the VAF percentages of both the Cart estimated Process Response and the Pendulum estimated Process Response is displayed for Sampling Instants: $k = 40 : 50$. Table 7.3 shows the same convergence as is presented in figure 7.9.

k =	40	41	42	43	44	45	46	47	48	49	50
VAF_1 in [%]	94.51	94.53	95.20	94.63	94.31	94.92	96.50	97.07	98.15	97.96	98.38
VAF_2 in [%]	0.078	0.66	30.74	2.84	37.65	48.99	93.99	98.40	98.46	97.27	98.15

Table 7.3: A quantitative model validation by using the VAF measure for each different realization

The VAF percentages converges to a VAF_1 percentage of 99.9967% for the Cart estimated Process Response and a VAF_2 percentage of 99.8222% for the Pendulum estimated Process Response. In figure 7.10 the Process Response corresponding to these VAF percentages is plotted:

$$\hat{y}_{1:K} = (\mu_A)_{\text{end}}(\mu_x)_{0:K} + (\mu_B)_{\text{end}}(u)_{0:K}$$

Figure 7.10 asserts that the equations derived in chapters 4 and 5 yield highly satisfactory and accurate results, if the correct data set is provided.

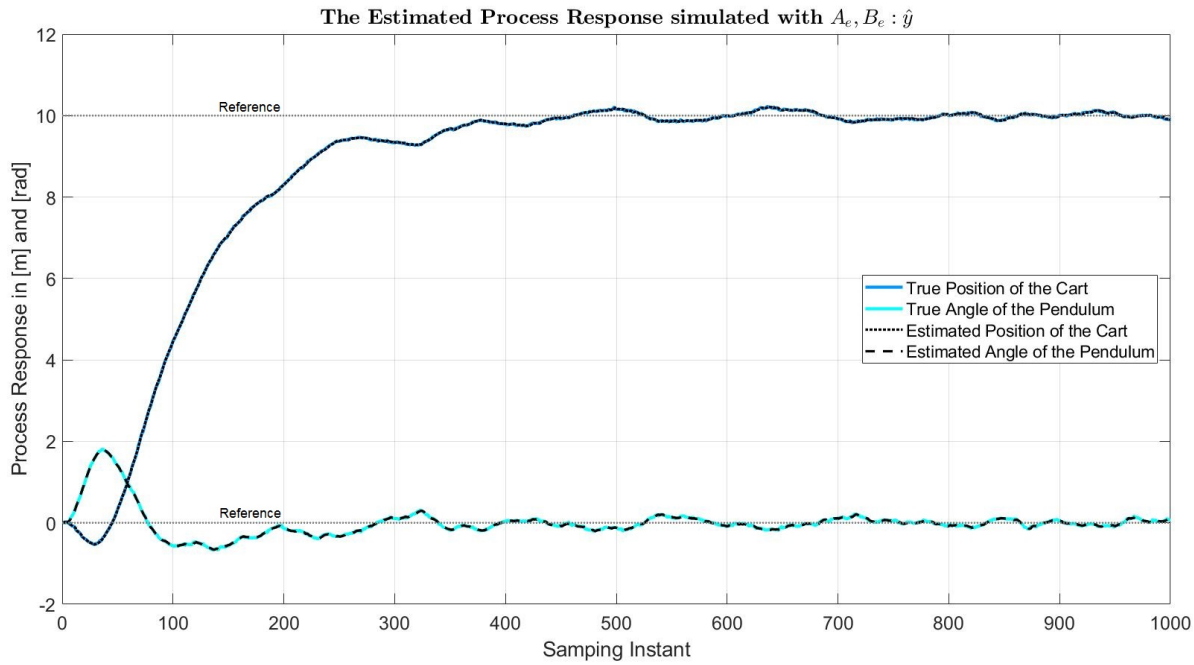


Figure 7.10: A comparison of the True Process Response with the estimated Process Response

In chapter 6 the Expectation and Maximization step where performed on various realizations of the simulated "True" Process Response. The "true" Process Response and, therefore as well, the estimated Process Response are for every realization, nearly, equivalent. Table 7.4 supports this statement. The VAF percentages only vary slightly.

Realization:	1	2	3	4	5	6
VAF_1 in [%]	99.9928	99.9932	99.9928	99.9926	99.9931	99.9927
VAF_2 in [%]	99.7595	99.7791	99.7485	99.7769	99.7801	99.7366

Table 7.4: A quantitative model validation by using the VAF measure for various realizations

As a final prove the Expectation and Maximization step to truly perform properly also under different circumstances, the steps are tested on two more noise Levels. The noise variances of equations 7.3 and 7.4 are selected as $\Sigma_w = \Sigma_z = 0.05$ and $\Sigma_w = \Sigma_z = 0.1$. The result is displayed in figure 7.11 and demonstrates the capability of the Free Energy Learning algorithm to cope with reasonably high noise levels. Besides the MPC Design to also perform well, even with the high noise levels, the Cart and Pendulum are brought to their desired Set-Points. However, it should be stressed the Expectation step to be in the possession of the noise variances: Σ_w and Σ_z, used to simulate the noise levels.

In the subsequent section, the Expectation and Maximization step will be incorporated into a Free Energy Learning algorithm where the correct model matrices are not provided. For the present, it is assumed that initial incorrect provided data: the model matrices will converge to the correct data as was observed in chapter 6. If this assumption holds the Free Energy algorithm could be a very attractive alternative to the currently applied System Identification techniques. Once convergence to the correct required data is established, the Free Energy algorithm is proven to yield very accurate results. The subsequent section will demonstrate whether the assumption is valid.

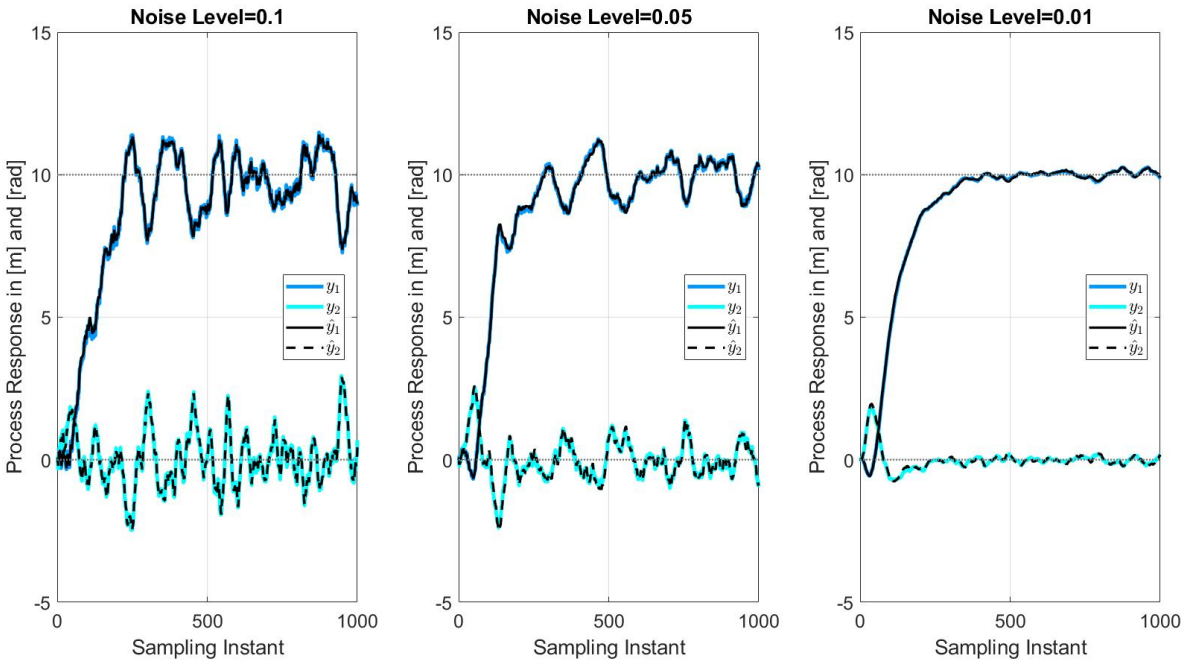


Figure 7.11: A comparison of the True Process Response with the estimated Process Response for different noise levels

7.4. THE FREE ENERGY LEARNING ALGORITHM

The Free Energy algorithm presented in table 7.4 performs the Expectation and Maximization step in an iterative manner. The Expectation step utilizes the previously estimated model matrices to estimate the state; the estimated state is thereafter utilized to estimate new model matrix estimates which then again are utilized in the subsequent iteration and so forth. A "True" Process Response and corresponding control trajectory are given as input to the algorithm just as initial model matrices. The Free Energy algorithm performs the Expectation and Maximization step in a simultaneous manner similar to the algorithm presented in table **??** from chapter 6. The difference is that in chapter 6 the MPC Design was also incorporated inside the iteration and, therefore, part of the simultaneous loop. For the Inverted Problem or multi-dimensional problems in general, this however entails problems with the Controllability of the Process and as a result control inputs and Process Responses to blow-up. Additional measures could prevent these issues but are outside of the scope of this thesis.

Algorithm 3: The multi-dimensional Free Energy Learning algorithm in an simultaneous manner with provided control trajectory and True Process Response

Step 0.1:	**Set initial conditions:**
	Set Control and Prediction horizons: N_p and N_c
	Set Initial model matrices: $A_0 \rightarrow \mu_A$ and $B_0 \rightarrow \mu_B$
	Set Initial state and state variance: $(\mu_x)_0$ and $(\Sigma_x)_0$
	Set desired set-points: R_s and R_u
Step 0.2:	**Perform Model Predictive Control:**
	$[u_{0:K}, y_{1:K}] =$ `MPCControl`$(A_d, B_d, C_d, R_s, R_u, N_p, N_c)$
Step 1:	**Perform Free Energy Learning algorithm:**
	Function $[\hat{y}_{1:K}, (\mu_x)_{0:K}, (\mu_A)_{0:K}, (\mu_B)_{0:K}] = \ldots$
	`FreeEnergyAlgorithm`$(A_0, B_0, (\mu_x)_0, (\Sigma_x)_0, u_{0:K}, y_{1:K})$
	For: $k = 1$ till selected # of Sampling instants: K:
	- Calculate the state estimation: $[(\mu_x)_k, (\Sigma_x)_k] = \ldots$
	`MPCFreeEnergyStateEst`$((\mu_A)_{k-1}, (\mu_B)_{k-1}, (\mu_x)_{k-1}, (\Sigma_x)_{k-1}, y_k, u_{k-1})$
	- Save the state estimation: $(\mu_x)_{0:K}(k) = (\mu_x)_k$
	- Calculate the model matrix estimations: $[(\mu_A)_k (\mu_B)_k] = \ldots$
	`FreeEnergyModelParametersEstMPC`$((\mu_A)_{k-1}, (\mu_B)_{k-1}, (\mu_x)_{k-1}, (\mu_x)_k, u_{k-1})$
	- Save the model matrix estimates: $(\mu_A)_{0:K}(k) = (\mu_A)_k$ and $(\mu_A)_{0:K}(k) = (\mu_B)_k$
	- Calculate the Process Response estimation: $\hat{y}_{k+1} = C_d \left((\mu_A)_k (\mu_x)_k + (\mu_B)_k u_k \right)$
	- Save the Process Response estimation: $\hat{y}_{1:K}(k+1) = \hat{y}_{k+1}$
	End
	End

Table 7.5: Algorithm 3: The multi-dimensional Free Energy Learning algorithm in an simultaneous manner with provided control trajectory and True Process Response

Initially the Free Energy algorithm was tested with completely random initial model matrices. This, however, did not yield to an accurate convergence. Substitution of the state statistical moments and model matrix statistical moments in the Free Energy certainly did not yield a minimum of the Free Energy landscape which ruled out that the algorithm had converged to a local minimum. More likely is there to be a correlation between different input and output signal which has not been acknowledge. Further research could consist out of further exploration of the multi-dimensional nature of the problem.

A considerably better result was achieved once the selected initial model matrices where still incorrect but much closer to the correct model matrices. This sort of selection of initial model matrices is sensible. Often, information is available, and it is not necessarily required to learn from scratch. The initial matrices where selected as the correct model matrices and an additional weighting factor: γ. This weighting factor determines how large the deviation from the actual model matrices is. The first weighting factor is selected as: $\gamma = 0.01$.

$$A_{\text{init}} = A_d + \gamma \cdot \texttt{randn}(4) \qquad\qquad B_{\text{init}} = B_d + \gamma \cdot \texttt{randn}(4,1) \tag{7.13}$$

The acquired state and model matrix estimations by the Free Energy algorithm are used to generate a prediction of the Process Response plotted in figure 7.12. The figure 7.12 is practically identical to figure 7.10.

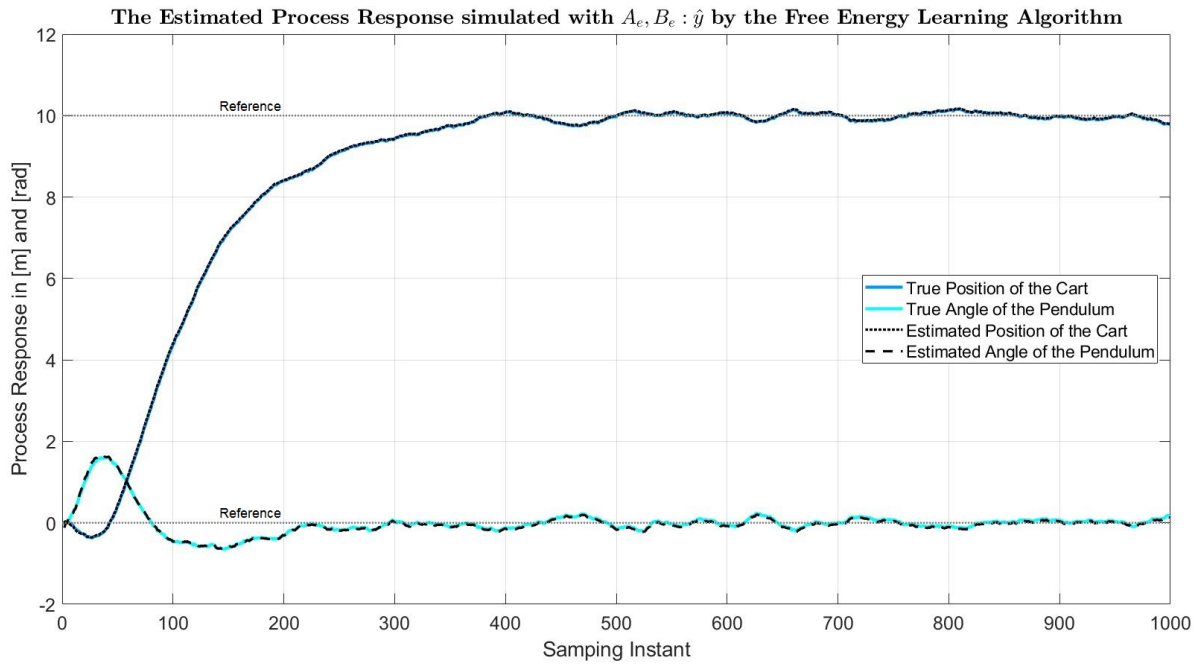


Figure 7.12: A comparison of the True Process Response with the estimated Process Response by the Free Energy Learning algorithm given incorrect initial model matrices

A numerical comparison tells there to be decay in performance with respect to the model matrix estimates found represented by equation 7.12. The difference between the estimated model matrices and the "True" Process matrices is $\approx 10^{-3}$ which can be considered as, only, a minor error but is one order higher as the previously found value. The error order, though, is one order lower than the weighting factor: γ, used in the initial model matrices represented by equation 7.13.

The corresponding model matrix estimations of the Free Energy algorithm are:

$$(\mu_A)_{\text{end}} = \begin{bmatrix} 1.0016 & 0.0160 & 0.0015 & 0.0005 \\ -0.0043 & 0.8690 & 0.0046 & 0.0028 \\ 0.0147 & 0.0046 & 0.9953 & 0.0110 \\ 0.0132 & 0.3042 & 0.2498 & 0.9953 \end{bmatrix} \qquad (\mu_B)_{\text{end}} = \begin{bmatrix} -0.0033 \\ 0.0849 \\ 0.0041 \\ -0.1883 \end{bmatrix} \tag{7.14}$$

It is furthermore worth noting the state variable estimations of the velocity of the Cart and the angular velocity of the Pendulum to be incorrect. On one hand this is the result of the increasing error in the model matrix estimations. On the other the error of the model matrices is a direct result of the error in the velocity state variable estimations. Multiple realizations of the script did show in some occasion an increase in the performance. The velocity state variable estimations where more accurate than the one displayed in figure 7.13. It seems the randomness of the noise to have a significant influence on the performance what it should not have.

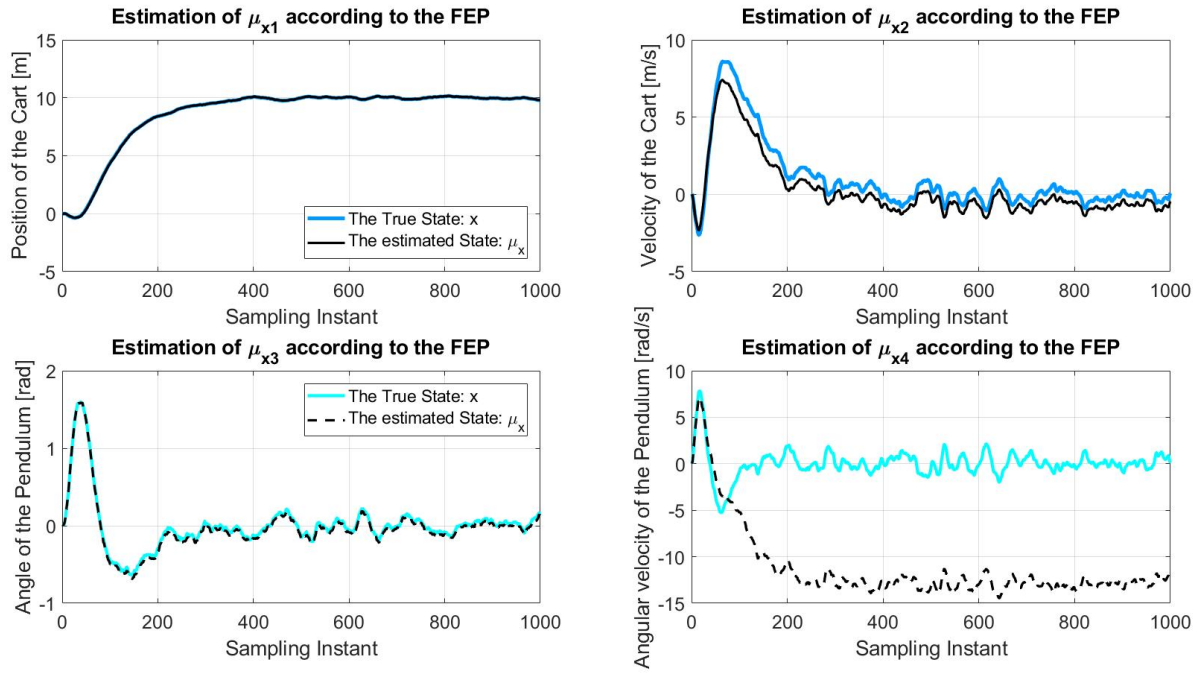


Figure 7.13: A comparison of the True Process state and state estimation by the Free Energy Learning algorithm with an error in the velocity state variable estimations

In addition to the previous made statement in figure 7.14 the state estimations are plotted for various weighting factors in the initial matrix where it is showed the more deviation is introduced the worse the state estimations become. The robustness of the Free Energy Learning algorithm is not yet what it should be. In the subsequent section it is assumed that with alterations of the implementation of the Free Energy algorithm after further explorations in existing theory, the Free Energy algorithm to be capable of achieving similar results as where presented in section: 1.3. The promise of that performance is compared to the performance of the well-known System Identification technique: Subspace Identification.

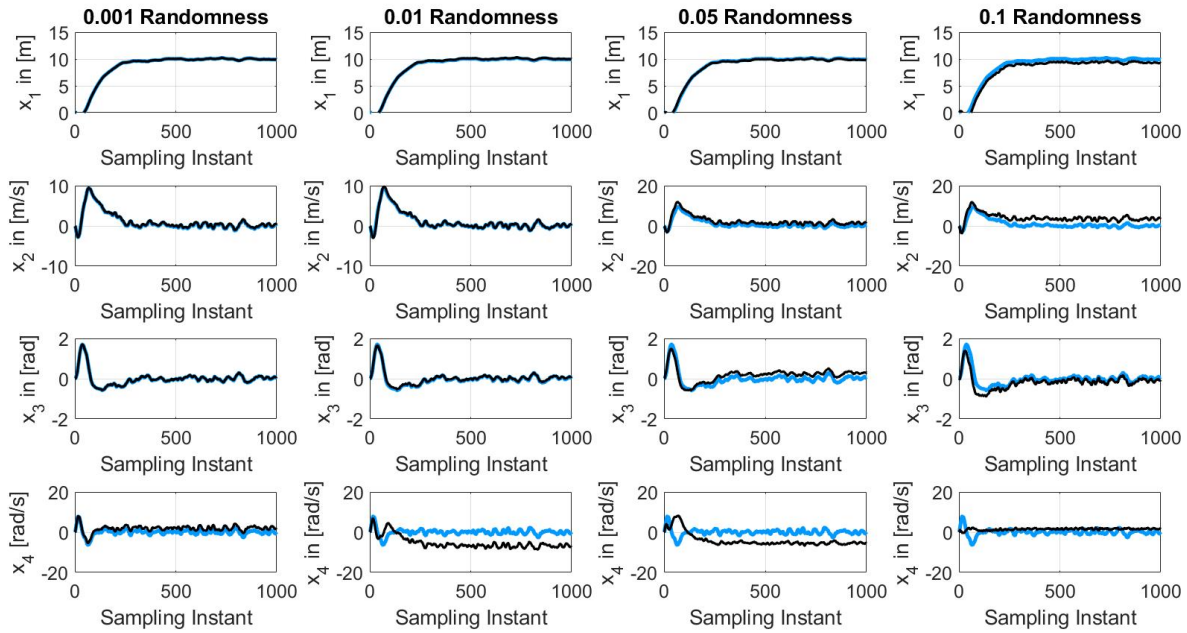


Figure 7.14: The decay of Performance when more deviations are introduced in the initial model matrices provided to the Free Energy Learning algorithm

7.5. VALIDATION OF THE FREE ENERGY LEARNING ALGORITHM BY COMPARISON WITH THE SI-TECHNIQUE: SUBSPACE IDENTIFICATION

To validate the Free Energy Learning algorithm besides the VAF percentage, a comparison is made with the similar Black-Box System Identification technique: Subspace Identification. Essentially both the System Identification techniques require only a set of control input and Process Response data. The primary difference between the Free Energy Learning algorithm and Subspace Identification algorithm is that the former is iterative and the latter is closed-form and single-step. This could yield to a more rapid computational time for the Subspace Identification algorithm but makes the technique less suitable to use in an adaptive manner. The model matrix estimation is performed offline rather than direct and online. The Free Energy Learning algorithm, conversely, possesses the proper structure and is capable of online estimation of the model matrices. The Subspace Identification algorithm is not sensitive to the specification of initial conditions; the algorithm does not require any initial conditions. The performance of the Free Energy Learning algorithm, conversely, is extremely dependent on the choice of initial conditions. And finally, it is worth to mention the Free Energy Learning algorithm to explicit state uncertainty in the form of variances of both the state and model matrices. The Subspace algorithm does not contain this explicit formulation of uncertainty [45].

The Subspace Identification algorithm requires the elements of the control input trajectory and Process Response to be stored into structured block Hankel Matrices. From these matrices, it is possible to retrieve Subspaces that are related to the state space model of the True Process [51].

$$Y_{i,s,N} = \begin{bmatrix} y_i & \cdots & y_{i+N-s} \\ y_{i+1} & \cdots & y_{i+N-s+1} \\ \vdots & \ddots & \cdots \\ y_{i+s-1} & \cdots & y_{i+N-1} \end{bmatrix} \qquad U_{i,s,N} = \begin{bmatrix} u_i & \cdots & u_{i+N-s} \\ u_{i+1} & \cdots & u_{i+N-s+1} \\ \vdots & \ddots & \cdots \\ u_{i+s-1} & \cdots & u_{i+N-1} \end{bmatrix} \tag{7.15}$$

The data equation $Y_{i,s,N} = \mathscr{O}_s X_{i,N} + \mathscr{T}_s U_{i,s,N}$ relates the Hankel matrices to matrices that can be constructed from the model matrices: (A_d, B_d, C_d, D_d). The matrix: $\mathscr{O}_s$, is known as the Observability matrix. The column space of the Observability matrix can be used to estimate the model matrices: (A_d, C_d). The first step in retrieving the column space of the Observability matrix is to obtain an estimate of the matrix: T_s. By subtraction of the estimate, the data equation reduces to $Y_{i,s,N} = \mathscr{O}_s X_{i,N}$ where each column of $Y_{i,s,N}$ represents a linear combination of the columns of the Observability matrix. When the rank of the output data is equivalent to the rank of the Observability matrix the column space of $Y_{i,s,N}$ is, as well, equivalent to column space of the Observability matrix. The column space of $Y_{i,s,N}$ is found by taking a **S**ingular **V**alue **D**ecomposition of $Y_{i,s,N} = U_n \Sigma_n V_n^T$. Furthermore, it is known how the Observability matrix is defined resulting in the following equality:

$$U_n = \mathscr{O}_s T = \begin{bmatrix} CT \\ CT(T^{-1}AT) \\ \vdots \\ CT(T^{-1}AT)^{s-1}) \end{bmatrix} = \begin{bmatrix} C_T \\ C_T A_T \\ \vdots \\ C_T A_T^{s-1} \end{bmatrix} \tag{7.16}$$

Where T is an unknown similarity transformation. The equality 7.16 can be solved for the model matrices (A_d, C_d). The estimate of the matrix: T_s, can be found as a solution to a Least-Square problem. More efficient is to avoid the explicit calculation of the estimation and use the following RQ Factorization:

$$\begin{bmatrix} Y_{i,s,N} \\ U_{i,s,N} \end{bmatrix} = \begin{bmatrix} R_{11} & 0 & 0 \\ R_{21} & R_{22} & 0 \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} \tag{7.17}$$

The column space of R_{22} is in a relatively similar fashion but in a more advanced manner related to the column space of $Y_{i,s,N}$, the column space of $Y_{i,s,N}$ times a orthogonal projection onto the column space of $U_{i,s,N}$ to be more precisely. The SVD of $R_{22} = U_n \Sigma_n V_n^T$ can, therefore, be used to calculate the similarity transforms estimates of the A and C model matrix: A_T and C_T. For this particular problem the C model matrix represents a unity state to output transition of the two state variable equal to the Process Response and, therefore, does not have to be estimated. For a more in-depth discussion of the theory one can read the following reference [51]. The book: [51], has a accompanying MATLAB LTI System Identification Toolbox [10]. The functions: `dac2bd` and `ltiitr`, from the Toolbox together with the estimation of the A model matrix are used to obtain an estimation of the model matrix B. The model matrix estimations by the Subspace algorithm are used to generate a prediction of the Process Response plotted in figure 7.12. From the plots displayed in figure 7.15 can be concluded that the Subspace algorithm results in an accurate estimate for the position of the Cart Process Response but an inaccurate estimate for the angle of the Pendulum Process Response. The same conclusion can be drawn from the accompanying VAF percentages where the VAF percentage corresponding to the position of the cart is equivalent to: 99.9557% and the VAF percentage corresponding to the angle of the Pendulum is equivalent to $-1.5621 \cdot 10^{-3}$%. The difference between the estimated model matrices and the "True" Process matrices is ≈ 0.05 which can be considered as a, fairly, high error in comparison to the errors present in the results of the Free Energy Learning algorithm. The inaccuracy of the estimates is presumably, due to that the currently implemented Subspace algorithm known as the MOESP algorithm not to consider the presence of any noises. An alteration of the implementation should provide better results and is considered next. The abbreviation MOESP stands for **M**ultivariable **O**utput **E**rror **S**tates**P**ace.

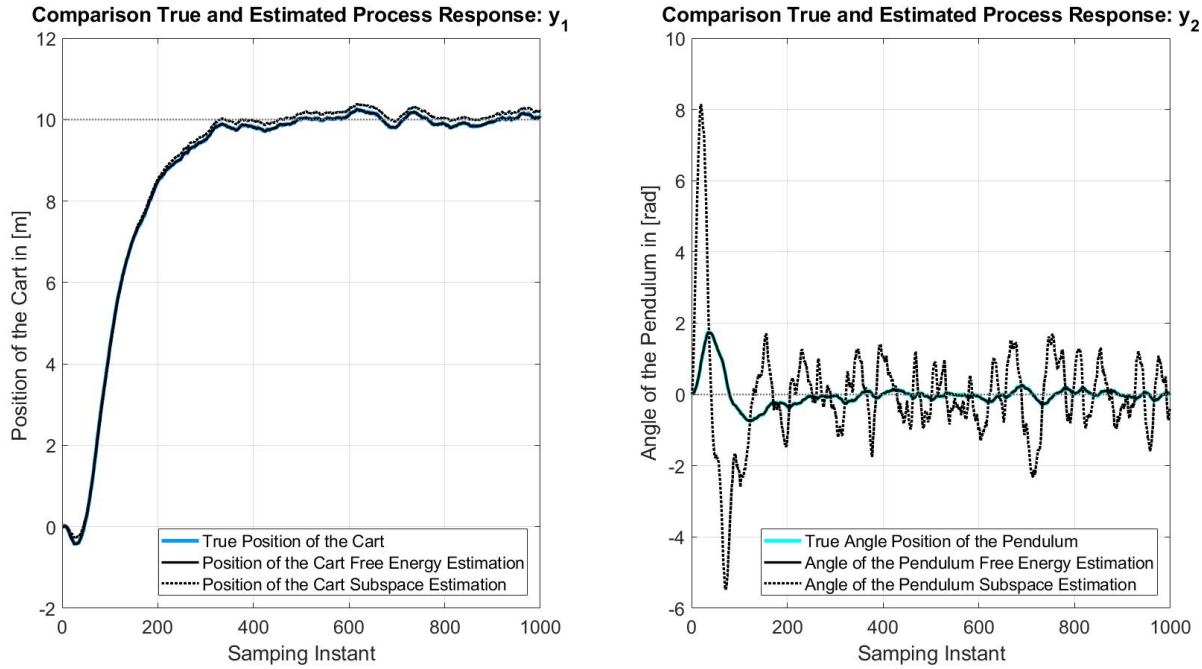


Figure 7.15: A comparison of the Subspace Identification estimated Process Response by the MOESP algorithm to the True Process Response and estimated Process Response by the Free Energy Learning algorithm

The first alteration is to acknowledge the presence of a noise component in the data equation. The block Hankel matrices are related through the following data equation:

$$Y_{i,s,N} = O_s X_{i,N} + \mathscr{T}_s U_{i,s,N} + \mathscr{S}_s E_{i,s,N} \tag{7.18}$$

To deal with the noise component an instrumental-variable approach is introduced.The use of the instrumental-variable matrix: $Z_n = \begin{bmatrix} U_{0,s,N} & Y_{i,s,N} \end{bmatrix}^T$, serves as the basis of the so-called PO-MOESP method where PO stands for **P**ast **O**utput. Once more, for an in-depth discussion of the PO-MOESP algorithm is referred to the reference: [51]. By the introduction of the instrumental-variable matrix: Z_n, the RQ Factorization of equation 7.17 is extended to:

$$\begin{bmatrix} U_{s,s,N} \\ \begin{bmatrix} U_{0,s,N} \\ Y_{0,s,N} \end{bmatrix} \\ Y_{s,s,N} \end{bmatrix} = \begin{bmatrix} R_{11} & 0 & 0 \\ R_{21} & R_{22} & 0 \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} \tag{7.19}$$

Instead of R_{22} the column space can be computed by taking the Singular Value Decomposition of R_{32}. The acquired model matrix estimations by the PO-MOESP Subspace algorithm are used to generate a prediction of the Process Response plotted in figure 7.16. From the plots displayed in figure 7.16, can be concluded there to be a slight decrease in the accuracy of the estimate for the position of the Cart Process Response but an improvement in the accuracy of the estimate for the angle of the Pendulum Process Response. The same conclusion can be drawn from the accompanying VAF percentages where the VAF percentage corresponding to the position of the cart is equivalent to: 94.9884% and the VAF percentage corresponding to the angle of the Pendulum is equivalent to 54.8694%. The difference between the estimated model matrices and the "True" Process matrices remains the same value of ≈ 0.05. Which results in the conclusion the Free Energy Learning algorithm to outperform the Subspace algorithm. This is, however, not entirely a fair statement to make and will be discussed in the concluding section of this chapter.

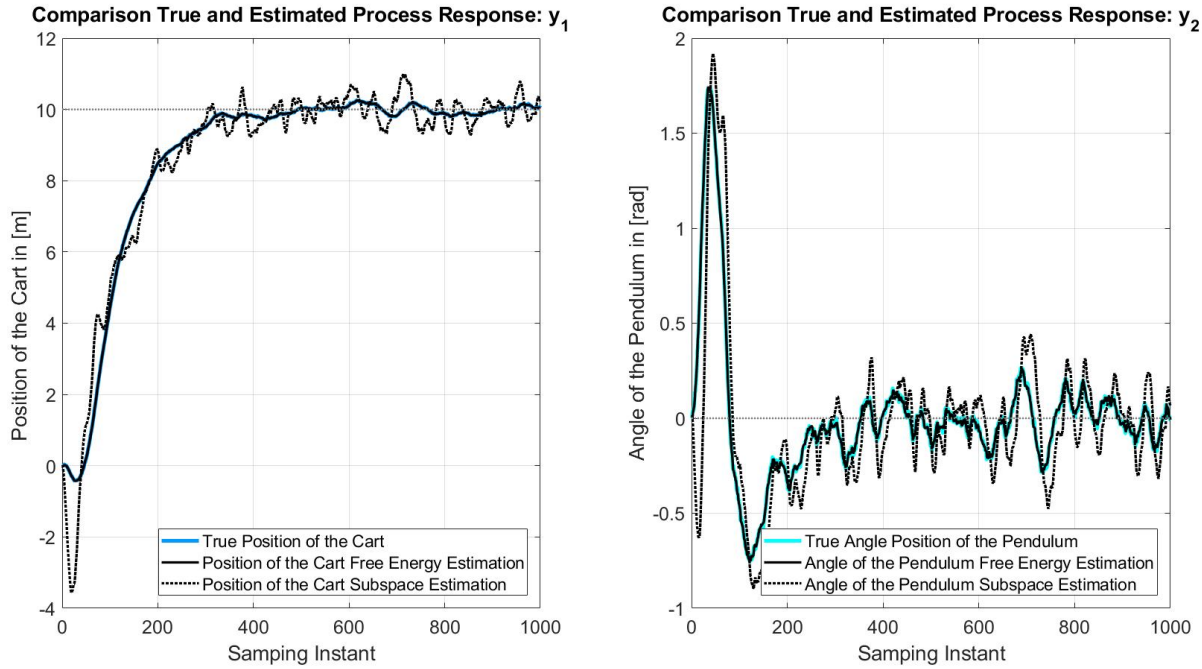


Figure 7.16: A comparison of the Subspace Identification estimated Process Response by the PO-MOESP algorithm to the True Process Response and estimated Process Response by the Free Energy Learning algorithm

7.6. DISCUSSION AND CONCLUSION

It is significant the Subspace Identification technique and Free Energy Principle are not so different in their theoretical concepts. Both algorithms adopt the concept of that once the state sequence is known the model matrix estimation is simply computed from a set of linear equations. In classical Identification methods the state sequence is not accessed until after the model matrix estimations are obtained. The state sequence purely provides functionality during the process of control [50]. In the Free Energy Learning algorithm and the Subspace algorithm, the state sequence is actively used during the process of System Identification although in different ways. Where in the Subspace algorithm the state sequence is determined from the projection of input and output data, in the Free Energy Learning algorithm the state sequence determination relies upon convergence of an iteration scheme. In this chapter the reliability of the current implementation of the Free Energy Learning algorithm appeared to be highly depending on the initially provided information. In this respect, the Subspace algorithm prevails and delivers consistent model matrix estimates. The current implementation of the Free Energy Learning algorithm is quite basic and there are many options to explore. Whether the Free Energy Learning algorithm will outperform the Subspace algorithm as it showed it could do once the required information was made available depends entirely on improvements of the Free Energy Learning algorithm. A more fair comparison can be made when both algorithms only have a correct set of control input and Process Response data at their disposal. This means that the noise variance currently provided to the Free Energy Learning algorithm has to be Learned as well. In the section: 5.4, is showed that the iteration loop given incorrect initial model matrices yields a deterioration in performance, while exploration of the dependencies of the different signals will presumably yield an improvement, it is likely that the addition of the learning of variances will yield a deterioration in the performance as well. Besides that the implementation of the Subspace algorithm is a Text Book example and, therefore, can be extended as well.

The parallels between Subspace Identification and the Free Energy Principle do not end with the similar adaptation of the state estimation. The Subspace algorithm uses past and future horizons clearly visible in the Hankel matrices represented by the equation 7.15. This could be translated to that for every time-step for a specific horizon past and future data is taken into account although in the Subspace algorithm specific time instances are not explicit and concealed in the mathematics. This corresponds somewhat to the Discrete-Time equivalent of the concept of generalized motion. It would be very interesting to incorporate either the Discrete-Time equivalent of the concept of generalized motion or past and future horizons in the current implemented Free Energy Learning algorithm and check whether there are indeed parallels. The Subspace algorithm remains, therefore, an interesting perspective in relation to the Free Energy algorithm.

8
DISCUSSION

This thesis serves to increase the understanding of the Free Energy principle and does so by evaluation of the Learning of Dynamical models according to the principles of Free Energy from a Control Theory perspective. This discussion will address some final thoughts on the overall thesis. For the more specific discussions is referred to the individual discussion at the end of each chapter.

The Free Energy principle explained by K. Friston is not straightforwardly applicable in Systems & Control settings. But also fields more familiar to the field of Neuroscience from which the Free Energy principle is derived experience difficulties. It is commonly agreed upon the Free Energy principle although containing appealing concepts to be hard to understand. A few examples of blog-posts recommended to read are [1], [32] and [42]. The theoretical framework of the Free Energy Principle becomes a lot more comprehensible when is realized the Free Energy principle to **heavily** rely upon the concepts of the Machine Learning method Variational Bayes which represents a Variational implementation of the Expectation Maximization methodology. The EM method employs a wide popularity in a broad variety of fields of applied statistics but is in the field of Control Engineering not yet widely appreciated [36]. For most Control Engineers unfamiliar with the EM methodology, it is difficult to separate the parts of the Free Energy Principle that are essentially concepts equivalent to the variational EM methodology. Take for example the derivation of the Free Energy presented in equations 3.5 - 3.7 from chapter 3. This derivation can exactly be found in relation to variational EM as for example in the references [4] and [5]. Only after a more devoted study into the EM algorithm, the link becomes more apparent. It is not necessarily true that K. Friston does not acknowledge this link however less explicit.

The are differences between the Free Energy principle and Variational EM methodology. The main differences identified are the Free Energy principle to consider the computation of the input trajectory to be a Free Energy optimization problem and the Free Energy principle to consider the concept of generalized motion. These are the two differences identified to substantiate the Free Energy principle algorithm from the variational EM algorithm. When the Learning part of the Free Energy principle is considered, only the difference with respect to generalized motion is relevant. In case of Learning, the Free Energy principle and Variational EM are similarly employed with the exception of generalized motion. In the chapters 4 and 5 this is proven specifically for the learning of linear models. The concept of generalized motion is in the chapters 4 and 5 not yet incorporated. The state estimation derived equation 4.29 from chapter 4 and the model matrix estimation derived equations 5.24 and 5.35 from chapter 5 are equivalent to the equations used in the variational EM algorithm. The equation used in the variational EM algorithm can for example be found in the references [4] and [31] were variational EM is applied on a linear problem. Both of these references were found quite early in the research but provide abstract formulations. The use of a simple optimization method is justified because a linear problem is considered. The use of a simple optimization method leads to closed form solutions. Rather than the recognition density to be an approximation -a close to exact solution- of the true posterior, an exact solution is found. In this case the Variational concept becomes redundant. The references [41], [24] and [23] address a regular form of the EM algorithm. The equations are equivalent to the derived equations in chapters 4 and 5.

The exact equivalence of the Free Energy Learning algorithm and EM algorithm was only identified in the final stages of the research that constituted this thesis. The implementations had a share in this accomplishment. The equations 6.19 and 6.20 are a function of one another. Given the information provided by the EM algorithm the dependency can be solved. The model matrix estimation equations then become only a function of the input, output and state. From the identified equivalence it is identified that a naive version of the EM algorithm is implemented [24]. This substantiates the statement made in chapter 7 that improvement of the Free Energy algorithm is required. A quicker discovery could have significantly improved the implementations.

9
RECOMMENDATIONS

In relation to the discussion chapter 8 and the individual discussion sections of each chapter 3, 4, 5, 6 and 7 the following recommendations for future work have been specified:

- The addition of the concept of generalized motion to the Free Energy Learning algorithm implemented in the chapters 6 and 6.4. Within the Free Energy principle, a Continuous-Time representation can be found. It would be very interesting to see whether the Discrete-Time equivalent of generalized motion bears any similarities with the horizon concept used in both Model Predictive Control (control horizon) and Subspace Identification (prediction horizon). Especially, because Subspace Identification and the EM algorithm both adopt the concept of that once the state sequence is known the model matrix estimation is simply computed from a set of linear equations.

- The consideration of the whole Free Energy framework meaning that the input trajectory is no longer generated by conventional control but according to the principles of Free Energy. In chapter 6 a control-loop was implemented making the performance of the Free Energy algorithm dependent on the performance of the control design. The use of a more advanced control design is identified as a possible improvement of the performance of the Free Energy algorithm. One could look into the already existing literature for more advanced MPC design. More interesting is to improve the Free Energy Learning algorithm by evaluation of the whole Free Energy principle applied.

- In the discussion, a link between the Free Energy Learning algorithm and the regular EM algorithm is identified. It was found that a naive version of the EM algorithm is implemented. It is recommended to further research the EM algorithm such that improvements of the implementations can be made. The Free Energy algorithm (i.e. the EM algorithm) promises to perform really well once convergence can be established. In chapter 7 it was currently impossible to create a control-loop. If the whole Free Energy principle would be considered this implementation requires an improvement. An improvement of the implementation will make it easier to make a fair comparison and evaluation of the performance. The algorithm has the promise to become a competitive algorithm for solving control problems.

- The incorporation of the hyper parameters or differently put the incorporation of the Prior probability distributions. In particular the incorporation of a computation of the noise variances which currently have been correctly provided to the algorithm. For real-time applications, these noise variances will probably not be constant and difficult to anticipate on.

- To try the implementations on actual set-ups as for example the real Pendulum-on-a-Cart set-up. It would be interesting to see whether the Free Energy Learning algorithm will indeed be capable to cope with model uncertainty. Furthermore, a real set-up can be used to improve the implementations. The implementations are currently only tested on known simulations. It is difficult to incorporate uncertainty or surprising behavior into a simulation.

10
CONCLUSION

The main research question of this thesis presented in the introductory chapter 2 is:

How does the Learning of dynamical models according to the principles of Free Energy relates to concepts from Control Theory?

In case of linearity and a Wiener Process assumption on the noise affecting the Process, the Free Energy Learning algorithm is equivalent to the Expectation Maximization algorithm. In the Expectation step the state is estimated by a Kalman filter. Once the state is known the estimation of the model parameters -in this case the model matrices A and B- becomes easy. In the maximization step the model matrices are estimated using a simple linear regression problem. Essentially, the Free Energy principle is all about the maximization of information. Once the knowledge becomes available, it is only easy to make predictions. In the EM algorithm the input is often assumed to be given together with the sensory data. In the Free Energy principle the state together with the input represents the complete data set required to compute the model parameters. The input trajectory is found by a similar optimization problem as the state or model parameters. If the Wiener Process assumption does not hold it is required to consider generalized motion. This has not been incorporated yet. Now that a basis implementation is provided it should be easier to incorporate the concepts of generalized motion. A possible relation with the prediction horizon used in the Subspace Identification method has been identified. Currently, a naive EM algorithm has been implemented as Free Energy Learning algorithm. The algorithm shows quite some potential. An initial test provided very accurate results however given a correct set of data. Once a convergence to this correct set of data has been established performance is assured. This requires improvements of the implementation. The exact equivalence with the EM algorithm was only identified in the final stages of the research that constitutes this thesis. Especially in the two-dimensional implementation improvements of the algorithm can provide additional insights.

A
MATLAB CODE

A.1. MATLAB GUIDE

All the self-written MATLAB code used in the chapters 6 and 7 is enclosed in this appendix. The scripts consist out of two main files and several functions used by the main file. A run of the one-dimensional main file provides the results presented in chapter 6. A full run of the script requires several minutes because some algorithms are performed multiple realizations. A quick check can be established by setting line 12 and 13 to be zero. There are no additional actions required. A run of the two-dimensional main file provides the result presented in chapter 7. The running time of this script is significantly less than the one-dimensional case.

A.2. ONE-DIMENSIONAL MATLAB SCRIPTS

```
1  %% Free Energy principle applied to the One-dimensional problem:
2  % Force driven mass
3  % Application of The Free Energy principle/
4  % Variational Bayes (Variational Expectation Maximization)
5  % Author: Julia Groeneweg, part of a graduation thesis, TU Delft (4076796)
6  % Faculty of 3ME, department of Mechanical Engineering and System & Control
7
8  clc; clear all; close all
9
10 % Full run requires about 10 minutes because of the multiple iterations
11 % For quick check set both to zero.
12 Simultaneous = 0; % Is part of twelve times iterations
13 MultipleRealizations = 0;
14
15 %% The Simulation
16 % Simulation of a Linear Dynamical System (LSD):
17 % Force driven mass with a constant drag Force (Resistance)
18 % The Generative process is a simulation, where the model matrices are
19 % derived from First Principles and perturbed by noise. The data generated
20 % from the generative process: the simulation is used for the
21 % implementation of the Free-Energy principle (Filtering in combination
22 % with System Identification part)
23
24 % Parameters (selected to reprent a car problem):
25 m = 1000;      % mass in [kg]
26 d = 10;        % drag-coefficient []
27
28 % The noise covariances, arbitrarely selected, check what the effect is
29 % once chosen smaller or larger:
30 Q = 0.01;      % Process Noise variance    var = sigma^2
31 R = 0.01;      % Measurement Noise variance
32
33 % State space representation, derived from first principles:
34 Ac = [-d/m]; Bc = [1/m]; Cc = 1; Dc = [0];
35
36 % Conversion from Continious-Time to Discrete-Time:
```

```
SysC   = ss(Ac,Bc,Cc,Dc); OmegaB = bandwidth(SysC); % sampling time should
% at least be twice as big as the bandwidth --> Shannon criterium

dt   = 0.1;               % Sampling time
SysD = c2d(SysC,dt);

Ad = SysD.A; Bd = SysD.B; Cd = SysD.C; Dd = SysD.D;

% Save useful data in data construct
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R');

% State space representation perturbed by noise: (Inclusion of noise model)
An = Ad; Bn = [Bd, 1, 0]; Cn = Cd; Dn = [Dd, 0, 1];

inputs  = {'Force'; 'w on x1'; 'z on y1'}; outputs = {'Vel Mass'};

SysN    = ss(An,Bn,Cn,Dn,dt, 'inputname',inputs, 'outputname',outputs);
set(SysN, 'StateName', {'Vel Mass'})

% The simulation time:
t = 0:dt:50;

% Generate an arbitrary input: a step response:
u_step = ones(length(t),1);
w      = randn(length(t),1)*sqrt(Q);          % Process noise
z      = randn(length(t),1)*sqrt(R);          % Measurement noise

% simulate the response: (Response is highly unstable)
[ys,ts,xs] = lsim(SysN,[u_step w z],t);

%% Model Predictive Conrol
% Control input trajectory is generated with known model matrices, the correct
% model matrices are provided, only a single iteration to bring the process
% to the desired reference.

% Set the horizons Nc<Np:
Np = 100; % Prediction horizon Np = 2 * int(2*Omegas/OmegaB) \approx 40, but
          % by increasing Np the stability is improved.
Nc = 2;   % Control horizon n ≤ Nc ≤ Np/2

% Augmented state-space, original plant model is augmented with
% integrators: (Subscript is a)
[SysA] = AugmentedSS(SysD);

% Calculation of the F and Phi matrices:
[PhiPhi, PhiF, PhiR] = MPCmatrices(SysA, Np, Nc);

% Number of Sampling instants:
Nsim = length(t);

% Selection of Set-point and Input limitations:
rs = 15; % Set-point, corresponds to 50 km/h = approx 15 m/s
Rs = rs*ones(Nsim,1);
ru = 0.1;
Ru = ru*eye(Nc,Nc); %Weight inputs, Input limitations

% Model Predictive Control option: with or without Free Energy state
% estimation:
StateEst = 0; % State estimation with Free Energy state estimation is off
              % first trial, only implementation of MPC design as presented
              % in literature to generate a System response used in the
```

```
                % Free Energy Principle

% Initial model matrices provided, SysD, are the correct matrices:
[u_mpc, y_mpc] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, StateEst, 0);

% Plot of the calculated control input and estimated output:
k=0:(Nsim-1);

f1 = figure;
subplot(211)
plot(k,y_mpc, 'k', 'LineWidth', 2); grid on; hold on;
title('The estimated Output Response by the MPC design')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]');
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
subplot(212)
plot(k,u_mpc, 'k', 'LineWidth', 2); grid on;
title('The generated Control Trajectory by the MPC design')
xlabel('Sampling Instant'); ylabel('Force in [N]');
supersizeme(f1, 1.5) % Enlarge fonts of the plot

% Apply the calculated control sequence on the generative process for
% multiple realizations of the stochastic process:
for run = 1:6

w = randn(length(t),1)*sqrt(Q);         % Process noise different every run
z = randn(length(t),1)*sqrt(R);         % Measurement noise

uN = [0,u_mpc]; yN = 0; xNN = 0;

    for i = 1:Nsim
        [yN(i), xN(i)] = simRealResponse(uN(i), xNN);
        xNN = xN(i);
    end

yNdata(:,:, run) = yN; xNdata(:,:, run) = xN;

if run == 1;   f2 = figure;  end

subplot(2,3,run)
plot(k,y_mpc, 'k', 'LineWidth', 1.5); grid on; hold on
plot(k,yN, 'k--', 'LineWidth', 2)
title(['Realization ' num2str(run)]);
hline           = refline([0 rs]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'The Noise-Free Response','The True Response'},'Location', 'southeast')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]'); ylim([0 20]);

MSE(run) = immse(yN, y_mpc); % mean((yN-y_mpc).^2)
end
supersizeme(f2, 1.4)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculates the Free Energy with respect to A and B for the state space
% representation derived from first principles and state sequence that
% should resemble the actual system, can be regarded as the True Free
% Energy:

Pp =0;
for i=1:Nsim
    Sigmaxp(:,:,i) = Ad*Pp*Ad + Q;
```

```
    K               = (Cd*Sigmaxp(:,:,i)*Cd+R)\Sigmaxp(:,:,i)*Cd;
    Pp              = (1-K*Cd)*Sigmaxp(:,:,i);
end

[FAtrue, FBtrue]    = FreeEnergy1D(ones(Nsim,1)* Ad, 0, ones(Nsim,1)* Bd, ...
                                   0, xN, Sigmaxp, u_mpc, Nsim);

f3 = figure;
subplot(121)
plot(k(2:Nsim),FAtrue(2:Nsim), 'k', 'LineWidth', 1.5); grid on;
xlabel('Sampling Instant'); ylabel('The Free Energy');
title('The True Free Energy with respect to A')
subplot(122)
plot(k(2:Nsim),FBtrue(2:Nsim), 'k', 'LineWidth', 1.5); grid on;
title('The True Free Energy with respect to B')
xlabel('Sampling Instant'); ylabel('The Free Energy');
supersizeme(f3, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Some Lines to check the Free Energy:
AFreeTest = [0.001, 0.01, 0.1, Ad, 2, 5];

for run = 1:6
Pp =0;
for i=1:Nsim
    Sigmaxp(:,:,i) = Ad*Pp*Ad + Q;
    K              = (Cd*Sigmaxp(:,:,i)*Cd+R)\Sigmaxp(:,:,i)*Cd;
    Pp             = (1-K*Cd)*Sigmaxp(:,:,i);
end

[FAtruetest, FBtruetest] = FreeEnergy(ones(Nsim,1)* AFreeTest(run), 0, ...
                           ones(Nsim,1)*Bd , 0, xN, Sigmaxp, u_mpc, Nsim);

FAtruetest250(run,:) = FAtruetest(250);
FBtruetest250(run,:) = FBtruetest(250);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Save useful data in data construct
% save variable in the output.mat file
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sys = load('data.mat');

%% The Free Energy principle
% Using the data-generated output where the Model Predicitive Controller
% does not uses the estimated Model matrices, yet.

% State estimation according to the Free Energy principle: (Kalman Filter)
% Initial conditions:
x0 = 0; P0 = 0;

[mu_x, Sigma_x, Sigma_x_prior] = FreeEnergyStateEst(sys, yN, x0, P0);

f4 = figure;
subplot(211)
plot(k,xN, 'c', 'LineWidth', 3); grid on; hold on;
plot(k,mu_x, 'k',  'LineWidth', 1.5)
title('The State estimation according to the Free Energy principle')
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'The True State: x','The estimated State: \mu_x'}, ...
```

```
        'Location', 'southeast')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]');
subplot(212)
plot(k,xN, 'c', 'LineWidth', 2); grid on; hold on
plot(k,mu_x, 'k',  'LineWidth', 1.5); title('Zoomed-in')
hline = refline([0 rs]); hline.Color  = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'The True State: x','The estimated State: \mu_x'}, ...
        'Location', 'southeast')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]');
axis([150 500 13 17])
supersizeme(f4, 1.5)

% Learning the Model matrices according to the Free Energy principle:
% Initial conditions, the hyper parameters are neglected
Ealpha = 0; Ebeta  = 0;

[muA, SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst1D(sys,yN, mu_x, ...
                              Sigma_x, Sigma_x_prior, Ealpha, Ebeta);

% Mean A and B are the estimates of model matrices A and B:
Ae = muA(end); Be = muB(end);

[Fa2, Fb2]   = FreeEnergy(ones(Nsim,1)* Ae, SigmaA, ...
                          ones(Nsim,1)* Be, SigmaB, ...
                          mu_x, Sigma_x_prior, u_mpc, Nsim);

f5 = figure;
subplot(121)
plot(k(2:Nsim), FAtrue(2:Nsim), 'c', 'LineWidth', 2)'; hold on; grid on;
plot(k(2:Nsim), Fa2(2:Nsim), 'k', 'LineWidth', 2)
title('The Free Energy with respect to A')
xlabel('Sampling Instant'); ylabel('The Free Energy');
legend('The True Free Energy','The estimated Free Energy', 'Location', ...
       'southeast')
subplot(122)
plot(k(2:Nsim), FBtrue(2:Nsim), 'c', 'LineWidth', 2); hold on; grid on;
plot(k(2:Nsim), Fb2(2:Nsim), 'k', 'LineWidth', 2)
title('The Free Energy with respect to B')
xlabel('Sampling Instant'); ylabel('The Free Energy');
legend('The True Free Energy','The estimated Free Energy', 'Location', ...
       'southeast')
supersizeme(f5, 1.5)

% Mean A and B are the estimates of model matrices A and B:
% Calculates the estimation of the output given all estimates:
yhat = Ae.*(mu_x.') + Be.*(u_mpc);

% Measure of how good the model accounts for the true output, the variance
% acounted for:
vaf = (1 - var(yN-yhat)/var(yN))*100;

f6 = figure;
subplot(211)
plot(k,yN, 'c', 'LineWidth', 2); grid on; hold on;
plot(k,yhat, 'k--', 'LineWidth', 1.5);
title({'\textbf{The Estimated Process Response simulated with} A_e,B_e: \hat y'},'Interpreter'
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'The True Process Response: y', 'The Estimated Process' ...
'Response: \hat{y}'}, 'Interpreter','Latex', 'Location', 'southeast')
```

```
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]');
subplot(212)
plot(k,yN, 'c', 'LineWidth', 2); grid on; hold on;
plot(k,yhat, 'k--', 'LineWidth', 1.5)
title({'\textbf{Zoomed-In}'},'Interpreter','Latex')
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'The True Process Response: y', ...
        'The Estimated Process Response: \hat y'},'Interpreter', ...
        'Latex','Location', 'southeast')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]');
axis([150 500 13 17])
supersizeme(f6, 1.5)

% Multiple realizations, State and Model Matrices estimation:
for run = 1:6
  x0 = 0; P0 = 0; yNrun = yNdata(:,:,run);

  [muxrun, Sigmaxrun, Sigmaxpriorrun] = ...
                                  FreeEnergyStateEst(sys, yNrun, x0, P0);

  if run == 1; f8 = figure; end

  subplot(2,3,run)
  plot(k,xNdata(:,:,run), 'c', 'LineWidth', 3); grid on; hold on;
  plot(k,muxrun, 'k',  'LineWidth', 1.5)
  title(['Realization ' num2str(run)])
  hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
  hline.LineWidth = 1.5; hline.LineStyle = ':';
  legend({'x','\mu_x'}, 'Location', 'southeast')
  xlabel('Sampling Instant'); ylabel('Velocity in [m/s]')
  axis([0 500 0 20])

  Ealpha = 0; Ebeta  = 0;

  [muArun, SigmaArun, muBrun, SigmaBrun] = ...
          FreeEnergyModelParametersEst1D(sys,yNrun, muxrun, ...
                              Sigmaxrun, Sigmaxpriorrun, Ealpha, Ebeta);

  Ae = muArun(end); Be = muBrun(end);
  yhatrun = Ae.*(muxrun.') + Be.*(u_mpc); yhatdata(:,:,run) = yhatrun;
  vafrun(run) = (1 - var(yNrun-yhatrun)/var(yNrun))*100;
end
supersizeme(f8, 1.4)

save('StochasticRealization.mat','MSE', 'vafrun')

for run = 1:6
  if run == 1; f9 = figure; end

  subplot(2,3,run)
  plot(k,yNdata(:,:,run), 'c', 'LineWidth', 2); grid on; hold on
  plot(k,yhatdata(:,:,run), 'k--', 'LineWidth', 1.5)
  title(['Realization ' num2str(run)])
  hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
  hline.LineWidth = 1.5; hline.LineStyle = ':';
  legend({'y','\hat y'},'Interpreter','Latex','Location', 'southeast')
  xlabel('Sampling Instant'); ylabel('Velocity in [m/s]'); ylim([0 20]);
end
supersizeme(f9, 1.4)

```

```
% Corresponding Probability density functions: (Recognition Density qa and
% qb)
mu = muA(end); sigma = SigmaA(end);
xA = mu-(4*sigma):(sigma / 100): mu+(4*sigma); yA = normpdf(xA,mu,sigma);

mu = muB(end); sigma = SigmaA(end);
xB = mu-(4*sigma):(sigma / 100): mu+(4*sigma); yB = normpdf(xB,mu,sigma);

f10 = figure;
subplot(221)
plot(xA,yA, 'k', 'LineWidth', 1.5); grid on
title('Recognition Density q_A')
subplot(222)
plot(xB,yB, 'k', 'LineWidth', 1.5); grid on
title('Recognition Density q_B')
subplot(223)
line([Ad Ad],[0 max(yA)],'Color','c', 'LineWidth', 2); grid on
title('The True Posterior p_A'); xlim([0.998 0.9999])
subplot(224)
line([Bd Bd],[0 max(yB)],'Color', 'c', 'LineWidth', 2); grid on
title('The True Posterior p_B'); xlim([9.990e-05 9.9990e-05])
supersizeme(f10, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Algorithm in a Control-Loop
% Set the horizons Nc<Np:
Np = 100; Nc = 2;

% Select initial matrices, does not have to make any sense, After a finite
% number of iterations the initial matrices converge towards the actual
% model matrices, for the speed it is convenient to select the matrices
% with some tought (not completely random)
Ad0   = [0.2]; Bd0   = [0.1];

dt    = 0.1;
SysD0 = ss(Ad0,Bd0,Cd,Dd,dt);  %Convert in a Discrete-time State space

% Initial conditions for the state estimation:
x0 = 0; P0 = 0.5;

% Initial conditions for the iteration, the loop, in general does not need
% more than 20 iterations, but is dependent on the initial matrices.
% Iteration scheme stops when the Variance Accounted for is higher than 99:
iter = 0; imax = 20; vaf  = 0;

% The Model Predicitive Controller uses the state estimation according to
% the Free Energy Principle, therefore:
StateEst = 1;

while vaf ≤ 99.7 && iter ≤ imax
% Makes sure that the model matrices are not negative, which is not correct
% and results in outputs that blow up:
SysD0.B = abs(SysD0.B); SysD0.A = abs(SysD0.A);

% Control input is calculated given the initial matrix:

[u_mpc, y_mpc] = MPCcontrol(SysD0, Np, Nc, Rs, Ru, Nsim, StateEst, 0);

% Limitations of the control signal due to really off chosen initial
% matrices:
u_mpc(u_mpc > 2000)    = 2000; u_mpc(u_mpc < -2000)   = -2000;
```

```
403 u_mpc(isnan(u_mpc))     = 0; u_mpc(¬isfinite(u_mpc))= 0;
404
405 % Simulates the output with the given control input, used in the state and
406 % model matrices estimations:
407 uN = [0,u_mpc]; xN = 0;
408
409     for i = 1:Nsim
410         [yN(i), xNN(i)] = simRealResponse(uN(i), xN); xN = xNN(i);
411     end
412
413 % saves variable in the data1.mat file, used in the functions:
414 Ad = SysD0.A; Bd = SysD0.B; Cd = SysD0.C; Dd = SysD0.D;
415 save('data1.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
416 sys = load('data1.mat');
417
418 % Estimation of the state according to the Free Energy principle:
419 [mu_x, Sigma_x, Sigma_x_prior] = FreeEnergyStateEst(sys, yN, x0, P0);
420
421 x0 = mu_x(1); P0 = Sigma_x(1);
422
423 % Calculates the Free Energy with the estimated values, and stores for
424 % for every iteration :
425 if iter ≥ 1
426 [Fa, Fb] = FreeEnergy(muA, SigmaA, muB, SigmaB, mu_x, ...
427                                     Sigma_x_prior, u_mpc, Nsim);
428 FA(:,:,iter) = Fa; FB(:,:,iter) = Fb;
429 end
430
431 % Estimation of the model matrices according to the Free Energy principle:
432 [muA,SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst1D(sys,yN, mu_x, ...
433                                 Sigma_x, Sigma_x_prior, Ealpha, Ebeta);
434
435 % Mean A and B are the estimates of model matrices A and B:
436 Ae = muA(end); Be = muB(end);
437
438 % Safety measures, are usually not needed:
439 Ae(isnan(Ae))=0; Ae(¬isfinite(Ae))=0; Be(isnan(Be))=1; Be(¬isfinite(Be))=1;
440
441 % Change the old estimates for the new ones, if the VAF is high enough the
442 % iteration loop will stop:
443 SysD0.A = Ae(end); SysD0.B = Be(end);
444
445 % Calculates the estimation of the output given all estimates:
446 yhat = Ae.*(mu_x.') + Be.*(u_mpc);
447
448 % Measure of how good the model accounts for the true output, the variance
449 % acounted for, should converge:
450 vaf = (1 - (var(yN-yhat)/var(yN)))*100;
451
452 % Safety measures, are usually not needed:
453 vaf(isnan(vaf))      = 1; vaf(¬isfinite(vaf)) = 1; iter = iter + 1;
454
455 % Stores interesting values for every iteration:
456 checkvaf(:,iter) = vaf; ycheck(:,:,iter)    = yN;
457 Atable(iter) = Ae; Btable(iter) = Be;
458 end
459
460 save('Iterationdata.mat','Atable', 'Btable', 'checkvaf')
461
462 % Calculates the Last-ouput with the converged estimates:
463 % Makes sure that the model matrices are not negative, which is not correct
```

```
% and results in outputs that blow up:
SysD0.B = abs(SysD0.B); SysD0.A = abs(SysD0.A);

% Control input is calculated given the initial matrix:
[u_mpc, y_mpc] = MPCcontrol(SysD0, Np, Nc, Rs, Ru, Nsim, StateEst, 0);

% Simulates the output with the given control input, used in the state and
% model matrices estimations:
uN = [0,u_mpc]; xN = 0;

    for i = 1:Nsim
        [yN(i), xNN(i)] = simRealResponse(uN(i), xN); xN = xNN(i);
    end

[mu_x, Sigma_x, Sigma_x_prior] = FreeEnergyStateEst(sys, yN, x0, P0);
[Fa, Fb] = FreeEnergy(muA, SigmaA, muB, SigmaB, mu_x, ...
                                Sigma_x_prior, u_mpc, Nsim);
FA(:,:,iter) = Fa;  FB(:,:,iter) = Fb;

% Finds the iterations where the VAF makes senses, values between 0 and
% 100 (Percentage), used in further plotting of the results:
ee = 1: iter; cc = checkvaf ≥ 0; dd = [];

 for i = ee
    if cc(i) == 1; dd = [dd, i]; end
 end

% Specifies grayscale for the plots (Lay-out):
colorVec = flipud(gray(iter));

% Bar-plot which displays the Variance Accounted For:
% (Only the iterations which make sense)
plotVAF = [];

for i = dd
    plotVAF = [plotVAF, checkvaf(i)];
end

figure
bar(dd, plotVAF, 'k'); grid on
title('The Variance Accounted For')
xlabel('Epochs'); ylabel('Percentage in [%]'); ylim([0 100])

% Convergence of the controlled output, again only the outputs
% corresponding to iterations with a normal Variance Accounted for are
% displayed:

f11 = figure; j = 1;
for i = dd(2:end)
    plot(k, ycheck(:,:,i), 'Color', colorVec(i,:), 'LineWidth', 2)
    hold on; grid on; legendInfo{j} = ['Epoch = ' num2str(i)-1]; j = j+1;
end
plot(k, yN, 'c', 'LineWidth', 2.5)
title('The True Process Response for selected Learning Epochs')
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]'); ylim([0 25])
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legendInfo{j} = ['Final Epoch'];
legend(legendInfo, 'Location', 'southeast')
supersizeme(f11, 1.5)

```

```
clear legendInfo
% The Free Energy calculations that belong to the iterations, once again,
% thate made sense:

f12 = figure; j = 1;
subplot(121)
for i = dd
plot(k(2:Nsim), FA(:,2:Nsim,i), 'Color',colorVec(i,:), 'LineWidth', 2)
hold on; grid on; legendInfo{j} = ['Epoch = ' num2str(i)]; j = j + 1;
end
plot(k(2:Nsim), FAtrue(2:Nsim), 'c', 'LineWidth', 2)
title('The Free Energy with respect to A')
xlabel('Sampling Instant'); ylabel('The Free Energy')
subplot(122)
for i = dd
plot(k(2:Nsim), FB(:,2:Nsim,i), 'Color',colorVec(i,:), 'LineWidth', 2)
hold on; grid on;
end
plot(k(2:Nsim), FBtrue(2:Nsim), 'c', 'LineWidth', 2)
title('The Free Energy with respect to B')
xlabel('Sampling Instant'); ylabel('The Free Energy')
legendInfo{j} = ['True Free Energy'];
legend(legendInfo, 'Location', 'southeast')
supersizeme(f12, 1.5)

% Quick test for the Free Energy
for kk = 1: iter
    FreeEnergyRuntestA(kk) = sum(FA(:,:,kk));
    FreeEnergyRuntestB(kk) = sum(FB(:,:,kk));
end

FreeEnergyRuntestA(kk+1) = sum(Fa2);
FreeEnergyRuntestB(kk+1) = sum(Fb2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% State Estimation and Learning inside the MPC Design
if Simultaneous == 1;
% Up till now the learning is done outside of the MPC design, this
% corresponds to batch Learning oppose to simultaneous Learning. The
% following lines of script considers simultaneos Learning more suitable
% for Real-Time Applications. Second Model-Control algorithm.

% Set Predictive and Control Horizon:
Nc = 2; Np = 100;

for run = 1:12
Rs = 15; Ru = 0.1;
% MPC design with Laguerre functions.
[u_Learning, y_Learning, yNLearning, SysMPC]  ...
                              =MPCcontrolFullLaguerre(Np, Nc, Rs, Ru, Nsim);

SysMPCdata(:,:,run) = SysMPC;
vafsim(run) = (1 - (var(yNLearning-(y_Learning.'))/var(yNLearning)))*100;

if run == 1; f13 = figure; end

subplot(3,4,run)
plot(k,yNLearning, 'k', 'LineWidth', 2); grid on; hold on
title(['Realization ' num2str(run)])
hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Sampling Instant'); ylabel('Velocity in [m/s]'); axis([0 500 0 25])
```

```
end
supersizeme(f13, 1.5)

end

%% Check realizations Control-Loop
if MultipleRealizations == 1;
% Runs the first model-Control Algorithm for twelve times to check wheter
% serval realizations yield to similar results
Atablerun = zeros(12,10); Btablerun = zeros(12,10);
checkvafrun = zeros(12,10);

for run= 1:12
clear legendInfo

% Set the horizons Nc<Np:
Np = 100; Nc = 4;

% Selection of Set-point and Input limitations:
rs = 15; % Set-point, corresponds to 50 km/h = approx 15 m/s
Rs = rs*ones(Nsim,1);
ru = 0.1;
Ru = ru*eye(Nc,Nc); %Weight inputs, Input limitations

% Select initial matrices, does not have to make any sense, After a finite
% number of iterations the initial matrices converge towards the actual
% model matrices, for the speed it is convenient to select the matrices
% with some tought (not completely random)
Ad0   = [0.2]; Bd0   = [0.1];

dt    = 0.1;
SysD0 = ss(Ad0,Bd0,Cd,Dd,dt);  %Convert in a Discrete-time State space

% Initial conditions for the state estimation:
x0 = 0; P0 = 0.5;

% Initial conditions for the iteration, the loop, in general does not need
% more than 20 iterations, but is dependent on the initial matrices.
% Iteration scheme stops when the Variance Accounted for is higher than 99:
iter = 0; imax = 20; vaf  = 0;

% The Model Predicitive Controller uses the state estimation according to
% the Free Energy Principle, therefore:
StateEst = 1;

while vaf ≤ 99 && iter ≤ imax
% Makes sure that the model matrices are not negative, which is not correct
% and results in outputs that blow up:
SysD0.B = abs(SysD0.B);
SysD0.A = abs(SysD0.A);

% Control input is calculated given the initial matrix:
[u_mpc, y_mpc] = MPCcontrol(SysD0, Np, Nc, Rs, Ru, Nsim, StateEst, 0);

% Limitations of the control signal due to really off chosen initial
% matrices:
u_mpc(u_mpc > 2000)    = 2000; u_mpc(u_mpc < -2000)   = -2000;
u_mpc(isnan(u_mpc))    = 0; u_mpc(¬isfinite(u_mpc))= 0;

% Simulates the output with the given control input, used in the state and
% model matrices estimations:
```

```
uN = [0,u_mpc]; xN = 0;

    for i = 1:Nsim
        [yN(i), xNN(i)] = simRealResponse(uN(i), xN); xN = xNN(i);
    end

% saves variable in the data1.mat file, used in the functions:
Ad = SysD0.A; Bd = SysD0.B; Cd = SysD0.C; Dd = SysD0.D;
save('data1.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sys = load('data1.mat');

% Estimation of the state according to the Free Energy principle:
[mu_x, Sigma_x, Sigma_x_prior] = FreeEnergyStateEst(sys, yN, x0, P0);

%x0 = mu_x(1); P0 = Sigma_x(1);

% Estimation of the model matrices according to the Free Energy principle:
[muA,SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst1D(sys,yN, mu_x, ...
                              Sigma_x, Sigma_x_prior, Ealpha, Ebeta);

% Mean A and B are the estimates of model matrices A and B:
Ae = muA(end); Be = muB(end);

% Safety measures, are usually not needed:
Ae(isnan(Ae))=0; Ae(¬isfinite(Ae))=0; Be(isnan(Be))=1; Be(¬isfinite(Be))=1;

% Change the old estimates for the new ones, if the VAF is high enough the
% iteration loop will stop:
SysD0.A = Ae(end); SysD0.B = Be(end);

% Calculates the estimation of the output given all estimates:
yhat = Ae.*(mu_x.') + Be.*(u_mpc);

% Measure of how good the model accounts for the true output, the variance
% acounted for, should converge:
vaf = (1 - (var(yN-yhat)/var(yN)))*100;

% Safety measures, are usually not needed:
vaf(isnan(vaf))     = 1; vaf(¬isfinite(vaf)) = 1; iter = iter + 1;

% Stores interesting values for every iteration:
checkvaf(:,iter) = vaf; ycheck(:,:,iter)   = yN;
Atable(iter) = Ae; Btable(iter) = Be;
end

% Calculates the Last-ouput with the converged estimates:
% Makes sure that the model matrices are not negative, which is not correct
% and results in outputs that blow up:
SysD0.B = abs(SysD0.B); SysD0.A = abs(SysD0.A);

% Control input is calculated given the initial matrix:
[u_mpc, y_mpc] = MPCcontrol(SysD0, Np, Nc, Rs, Ru, Nsim, StateEst, 0);

% Simulates the output with the given control input, used in the state and
% model matrices estimations:
uN = [0,u_mpc]; xN = 0;

    for i = 1:Nsim
        [yN(i), xNN(i)] = simRealResponse(uN(i), xN); xN = xNN(i);
    end

```

```
% Finds the iterations where the VAF makes senses, values between 0 and
% 100 (Percentage), used in further plotting of the results:
ee = 1: iter; cc = checkvaf ≥ 0; dd = [];

 for i = ee
    if cc(i) == 1; dd = [dd, i]; end
 end

% Specifies grayscale for the plots (Lay-out):
colorVec = gray(iter);

% Bar-plot which displays the Variance Accounted For:
% (Only the iterations which make sense)
plotVAF = [];

for i = dd; plotVAF = [plotVAF, checkvaf(i)]; end

if run == 1; f14 = figure; end

subplot(3,4, run)
bar(dd, plotVAF, 'k'); hold on; grid on
title(['Realization ' num2str(run)])
xlabel('Epochs'); ylabel('Percentage in [%]'); ylim([0 100])

Atablerun(run,1:size(Atable,2)) = Atable;
Btablerun(run,1:size(Btable,2)) = Btable;
checkvafrun(run,1:size(checkvaf,2)) = checkvaf;
Ynrun(:,:,run) = yN;

save('Iterationdatarun.mat','Atablerun', 'Btablerun', 'checkvafrun')
end
supersizeme(f14, 1.5)

for run = 1:12
    if run == 1; f15 = figure; end

    subplot(3,4, run)
    plot(k, Ynrun(:,:,run), 'k', 'LineWidth', 2 ); hold on; grid on
    title(['Realization ' num2str(run)])
    xlabel('Sampling Instants'); ylabel('Velocity in [m/s]')
    xlim([0 500]); ylim([0 20])
    hline = refline([0 rs]); hline.Color = [0.5 0.5 0.5];
    hline.LineWidth = 1.5; hline.LineStyle = ':';
end
supersizeme(f15, 1.5)
end
```

```
function [FA, FB] = FreeEnergy1D(muA, SigmaA, muB, SigmaB, mux, ...
                                 Sigmaxp, u, Nsim)
% Calculates the Free Energy
if SigmaA ==0
    for k = 2:Nsim
        Sxp = 1\Sigmaxp(:,:,k-1);
        Fa = -log(1/sqrt(2*pi*abs(Sigmaxp(:,:,k)))) ...
             -(1/2)*mux(k)*Sxp*muA(k)*mux(k-1) ...
             -(1/2)*mux(k-1)*muA(k)*Sxp*mux(k) ...
             +(1/2)*mux(k-1)*(muA(k)*muA(k))*Sxp*mux(k-1) ...
             +(1/2)*mux(k-1)*muA(k)*Sxp*muB(k)*u(k-1) ...
             +(1/2)*u(k-1)*muB(k)*Sxp*muA(k)*mux(k-1);
```

```
        FA(k) = Fa;

        Fb = -log(1/sqrt(2*pi*abs(Sigmaxp(:,:,k)))) ...
             -(1/2)*mux(k)*Sxp*muB(k)*u(k-1) ...
             +(1/2)*mux(k-1)*muA(k)*Sxp* muB(k)*u(k-1) ...
             -(1/2)*u(k-1)* muB(k)*Sxp* mux(k) ...
             +(1/2)*u(k-1)* muB(k)*Sxp* muA(k)*mux(k-1) ...
             +(1/2)*u(k-1)*(muB(k)*muB(k))*Sxp*u(k-1);
        FB(k) = Fb;
    end
else
    for k = 2:Nsim
        Sxp = 1\Sigmaxp(:,:,k-1);
        Fa = -log(1/sqrt(2*pi*abs(Sigmaxp(:,:,k)))) ...
             -(1/2)*mux(k)*Sxp*muA(k)*mux(k-1) ...
             -(1/2)*mux(k-1)*muA(k)*Sxp*mux(k) ...
             +(1/2)*mux(k-1)*(SigmaA(:,:,k)+ muA(k)*muA(k))*Sxp*mux(k-1) ...
             +(1/2)*mux(k-1)*muA(k)*Sxp*muB(k)*u(k-1) ...
             +(1/2)*u(k-1)*muB(k)*Sxp*muA(k)*mux(k-1); ...
             %-log(1/sqrt(2*pi*abs(SigmaA(:,:,k))));
        FA(k) = Fa;

        Fb = -log(1/sqrt(2*pi*abs(Sigmaxp(:,:,k)))) ...
             -(1/2)*mux(k)*Sxp*muB(k)*u(k-1) ...
             +(1/2)*mux(k-1)*muA(k)*Sxp* muB(k)*u(k-1) ...
             -(1/2)*u(k-1)* muB(k)*Sxp* mux(k) ...
             +(1/2)*u(k-1)* muB(k)*Sxp* muA(k)*mux(k-1) ...
             +(1/2)*u(k-1)*(SigmaB(:,:,k)+ muB(k)*muB(k))*Sxp*u(k-1); ...
             %-log(1/sqrt(2*pi*abs(SigmaB(:,:,k))));
        FB(k) = Fb;
    end
end
```

```
function [muA, SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst1D(sys, y, mu_x, Sigma_x, Si

% The Control Input sequence:
u = sys.u_mpc;

% provided estimates/expectations of the model matrices, both are used in
% the estimation process of the other
B = sys.Bd; A = sys.Ad;

% Initial conditions
[Nsim,n] = size(mu_x);

%% The estimation of Model matrix A
% The variance:
Sigma_a_part = zeros(n);
    for k = 2:Nsim
     aaa = (Sigma_x_prior(:,:,k-1))\(mu_x(k-1,:)*mu_x(k-1,:));
     Sigma_a_part = Sigma_a_part + aaa;
     SigmaA(:,:,k) = inv(Ealpha + Sigma_a_part);
    end
% The Mean:
sumx0x1T = zeros(n); sumx0u1BT = zeros(n);
     for k = 2:Nsim
      x0x1T = (mu_x(k-1)* mu_x(k)); sumx0x1T = sumx0x1T + x0x1T;
      x0u1BT = (mu_x(k-1)*B*u(k-1)); sumx0u1BT = sumx0u1BT + x0u1BT;

```

```
27        ans = (Sigma_x_prior(:,:,k-1))\((sumx0x1T - sumx0u1BT));
28        muA(k,1) = SigmaA(:,:,k)*ans;
29       end
30  %% The estimation of Model matrix B
31  % The variance:
32  Sigma_b_part = zeros(n);
33      for k = 2:Nsim
34       bbb = (Sigma_x_prior(:,:,k-1))\(u(k-1)*u(k-1));
35       Sigma_b_part = Sigma_b_part + bbb;
36       SigmaB(:,:,k) = inv(Ebeta + Sigma_b_part);
37      end
38  sumu0x1T = zeros(1,n); sumu0x1AT = zeros(1,n);
39      for k = 2:Nsim
40       u0x1T = (u(k-1)* mu_x(k)); sumu0x1T = sumu0x1T + u0x1T;
41       u0x1AT = (u(k-1)*A*mu_x(k-1)); sumu0x1AT = sumu0x1AT + u0x1AT;
42
43       ans = (Sigma_x_prior(:,:,k-1))\((sumu0x1T - sumu0x1AT).');
44       muB(k,1) = SigmaB(:,:,k)*ans;
45      end
46  end
```

```
1   function [muA,SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst(sys, y, mu_x, Sigma_x, Sigma
2   % The Control Input sequence:
3   u = sys.u_mpc;
4
5   % provided estimates/expectations of the model matrices, both are used in
6   % the estimation process of the other
7   B = sys.Bd; A = sys.Ad;
8
9   % Initial conditions
10  [n,Nsim] = size(mu_x); p         = size(B, 2);
11
12  Ealpha    = Ealpha .* eye(n*n); Ebeta     = Ebeta .* eye(n*p);
13
14  %% The estimation of Model matrix A
15  % The variance:
16  Sigma_a_part = zeros(n*n);
17      for k = 2:Nsim
18       aaa = times(inv(Sigma_x_prior(:,:,k-1)),(mu_x(:,k-1).')*(mu_x(:,k-1)));
19       aaa = (kron(eye(n), aaa).');
20       Sigma_a_part = Sigma_a_part + aaa;
21       SigmaA(:,:,k) = inv(Ealpha + Sigma_a_part);
22      end
23  % The Mean:
24  sumx0x0T = zeros(n); sumx0x1T = zeros(n); sumx0u1BT = zeros(n);
25       for k = 2:Nsim
26        x0x0T = (mu_x(:,k-1)* (mu_x(:,k-1).')); sumx0x0T = sumx0x0T + x0x0T;
27        x0x1T = (mu_x(:,k-1)* (mu_x(:,k).')); sumx0x1T = sumx0x1T + x0x1T;
28        x0u1BT = (mu_x(:,k-1)*((B*u(k-1)).')); sumx0u1BT = sumx0u1BT + x0u1BT;
29
30        ans = ((sumx0x1T - sumx0u1BT).')*inv(sumx0x0T); muA(:,:,k) = ans;
31       end
32  %% The estimation of Model matrix B
33  % The variance:
34  Sigma_b_part = zeros(n);
35      for k = 2:Nsim
36       bbb = (u(k-1)*u(k-1))./ inv(Sigma_x_prior(:,:,k-1));
37       Sigma_b_part = Sigma_b_part + bbb;
38       SigmaB(:,:,k) = inv(Ebeta + Sigma_b_part);
```

```
    end
sumu0u0T = 0; sumu0x1T = zeros(1,n); sumu0x1AT = zeros(1,n);
    for k = 2:Nsim
     u0u0T = (u(k-1)*u(k-1)); sumu0u0T = sumu0u0T + u0u0T;
     u0x1T = (u(k-1)* (mu_x(:,k).')); sumu0x1T = sumu0x1T + u0x1T;
     u0x1AT = (u(k-1)* ((A*mu_x(:,k-1)).')); sumu0x1AT = sumu0x1AT + u0x1AT;

     ans = ((sumu0x1T - sumu0x1AT).')*inv(sumu0u0T); muB(:,:,k) = ans;
    end
end
```

A.3. TWO-DIMENSIONAL MATLAB SCRIPTS

```
%% Free Energy principle applied to the bench problem: Pendulum on a cart
% Application of The Free Energy principle/
% Variational Bayes (Variational Expectation Maximization)
% Author: Julia Groeneweg, part of a graduation thesis, TU Delft (4076796)
% Faculty of 3ME, department of Mechanical Engineering and System & Control
clc; clear all; close all
%% The Simulation
% Simulation of a Linear Dynamical System (LSD):
% Linearized pendulum on a cart
% The Generative process is a simulation, where the model matrices are
% derived from First Principles and perturbed by noise. The data generated
% from the generative process: the simulation is used for implementation
% of the Free-Energy principle (Filtering in combination with System
% Identification part)
% Parameters
M = 0.49;           % [kg] mass cart
m = 0.085;          % [kg] mass pendulum
% b = 0.1;          % [] Initial assumed damping
% Driving mechanism is DC-motor with following properties:
C  = 10.42;         % [] Viscous Damping (Higher than initial damping)
Im = 0.3678;        % [] Motor Inertia
Kv = 8.389;         % [] The gain conversion from voltage to Force
g = 9.81;           % [kg/m^2] gravity
l = 0.3;            % [m] length pendulum
% I = 0.006;        % [] Initial assumed Inertia, I = (1/3)*m*l^2
J = (1/3)*m*l^2;    % [] Inertia

% Time-vector
dt  = 0.01;         % Sample time [sec]
t   = 0:dt:10; t   = t.'; % time vector (-)

% Common factor, denominator:
ccc = (Im + M + m/4);
Ac = [ 0,               1,                                0,  0;
       0,          -C/ccc,              -(3*g*m)/(4*ccc),  0;
       0,               0,                                0,  1;
       0, (3*C)/(4*l*ccc), (3*(g + (3*g*m)/(4*ccc)))/(4*l), 0];
Bc = [                  0;
                    Kv/ccc;
                         0;
        -(3*Kv)/(4*l*ccc)];
Cc      = [1 0 0 0;
           0 0 1 0];
Dc      = [0; 0];

```

```
inputs  = {'Voltage'}; outputs = {'Pos Cart'; 'Pos Pendulum'};
SysC    = ss(Ac,Bc,Cc,Dc,'inputname',inputs, ...
             'outputname',outputs);
set(SysC, 'StateName', {'Pos Cart', 'Vel Cart', ...
          'Pos Pendulum', 'Vel Pendulum'})

% Conversion from Continious-Time to Discrete-Time
ds              = 0.01; % Samping time in [sec], sampling time should
% at least be twice as big as the bandwidth --> Shannon criterium
SysD            = c2d(SysC, ds); [Ad,Bd,Cd,Dd]   = ssdata(SysD);
% State space representation perturbed by noise: (Inclusion of noise model)
% Subscript is n
An =  SysD.A; Bn = [SysD.B eye(4) zeros(4,2)]; % x[k+1] = Ax[k]+Bu[k]+w[k]
% x[k+1] = Ax[k] + [B 1 0]*[u[k]; w[k]; z[k]]
Cn =  SysD.C; Dn = [SysD.D zeros(2,4) eye(2,2)]; % y[k+1] =Cx[k]+Du[k]+z[k]
% y[k+1]= Cx[k] + [D 0 1]*[u[k]; w[k]; z[k]]

inputs  = {'Voltage'; 'w on x1'; 'w on x2';'w on x3'; 'w on x4'; ...
           'z on y1'; 'z on y2'}; outputs = {'Pos Cart'; 'Pos Pendulum'};
SysN    = ss(An,Bn,Cn,Dn,ds, 'inputname',inputs, ...
             'outputname',outputs);
set(SysC, 'StateName', {'Pos Cart', 'Vel Cart', ...
          'Pos Pendulum', 'Vel Pendulum'})

% The Measurement and process variance:
Q = 0.001;       % Process noise variance
R = 0.001;       % Measurement noise variance

% Generate an arbitrary input: a step response.
u_step = ones(size(t));
w = randn(length(t),4)*sqrt(Q); % Process noise (Sqrt because std^2=var)
z = randn(length(t),2)*sqrt(R); % Measurement noise

% simulate the response (Response is highly unstable)
[ys,ts,xs] = lsim(SysN,[u_step w z],t);

step_info = lsiminfo(ys,t); cart_info = step_info(1);
pend_info = step_info(2)

% Model order, number of outputs, number of inputs
n = size(Ad,1); p = size(Bd,2); m = size(Cd,1);

%% Check for controllability
% Controllability is required in order for the design of MPC to access all
% dynamics modes in the system.
if rank(ctrb(SysN)) == length(eig(SysN))
    'Model is controllable'
else
    'Model is NOT controllable'
    return
end;

% Save useful data in data construct
% save variable in the output.mat file
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R')

%% Model Predictive Control
% Set the horizons Nc<Np
Np = 180; % Prediction horizon
Nc = 10;  % Control horizon

```

```
% Augmented state-space, original plant model is augmented with
% integrators. (Subscript is a)
[SysA] = AugmentedSS(SysD);
Aa = SysA.A; Ba = SysA.B; Ca = SysA.C; Da = SysA.D;

% The augmented state space:
inputs  = {'Delta Voltage'}; outputs = {'Pos Cart'; 'Pos Pendulum'};

SysA    = ss(Aa,Ba,Ca,Da,ds, 'inputname',inputs, ...
              'outputname',outputs);
set(SysA, 'StateName', {'Delta Pos Cart', 'Delta Vel Cart', ...
          'Delta Pos Pendulum', 'Delta Vel Pendulum', ...
          'Pos Cart','Pos Pendulum'})

      %% MPC using the Augmented State-Space and a simple optimization
% with respect to the cost function:
% J = (Rs -Y)^T(Rs-Y) + DeltaU^T Rbar DeltaU
% Receding horizon control --> In order to find the control law we need the
% matrices F and Phi: DeltaU = inv(PhiPhi^T + Rbar)Phi^T(Rs-F*x)
StateEst = 0;       % State Estimation with FreeEnergyStateEst script off
Laguerre = 0;       % MPC control with simple optimization

% Simulation time
Nsim=length(t)-1; %number of samples

% Determine set-point, or reference signal:
% Reference (Position of cart and pendulum [m] and [rad]
rs = [10;0]; Rs = [rs(1)*ones(Nsim,1),rs(2)*ones(Nsim,1)];
ru = 1; Ru = ru*eye(Nc,Nc); %Weight inputs

[u_mpc, y_mpc] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, StateEst, Laguerre);
k=1:Nsim; % Sampling instants

f1 = figure;
subplot(211)
plot(k,y_mpc(1,:), 'k', 'LineWidth', 2); hold on; grid on
plot(k,y_mpc(2,:), 'k--', 'LineWidth', 2)
title('The estimated Output Response by the MPC Design')
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
legend({'Position of the Cart','Angle of the Pendulum', },'Location', ...
        'east')
subplot(212)
plot(k,u_mpc, 'k', 'LineWidth', 2); grid on
xlabel('Sampling Instant'); ylabel('Control input in Voltage [V]')
title('The generated Control Trajectory by the MPC Design')
supersizeme(f1, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Save useful data in data construct
% save variable in the output.mat file
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sysdata = load('data.mat');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xNN = zeros(4,1); u_mpc=[0,u_mpc]; % x0 u0
    for i = 1:Nsim
        [yNN, xNN] = simRealResponse(u_mpc(i), xNN);
        xN(:,i)    = xNN; yN(:,i)    = yNN;
```

```
    end

f2 = figure;
subplot(211)
plot(k, yN(1,:), 'Color', [0 .6 1], 'LineWidth', 2); grid on; hold on
plot(k, yN(2,:), 'c', 'LineWidth', 2)
title('The True Process Response')
legend({'Position of the Cart','Angle of the Pendulum'},'Location', ...
        'northwest')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
subplot(212)
plot(k, yN(1,:), 'Color', [0 .6 1], 'LineWidth', 2.5); grid on; hold on
plot(k, yN(2,:), 'c', 'LineWidth', 2.5)
plot(k,y_mpc(1,:), 'k', 'LineWidth', 2)
plot(k,y_mpc(2,:), 'k--', 'LineWidth', 2)
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':'; axis([0 200 -20 20])
title('The Noise-Free Process Response compared to the True Process Response')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
legend({'True Position of the Cart','True Angle of the Pendulum', ...
        'Estimated Position of the Cart', 'Estimated Angle of the Pendulum'}, ...
        'Location', 'southeast')
supersizeme(f2, 1.5)

%% The Design of a MPC control scheme Using Laguerre Functions
StateEst = 0;       % State Estimation with FreeEnergyStateEst script off
Laguerre = 1;       % MPC control with Laguerre Functions

% Determine set-point, or reference signal:
% Reference (Position of cart and pendulum [m] and [rad]
rs = [10;0]; Rs = [rs(1)*ones(Nsim,1),rs(2)*ones(Nsim,1)];
ru = 0.1; Ru = diag(ru * ones(1,Nc));

[u_mpcL, y_mpcL] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, StateEst, Laguerre);

f3 = figure;
plot(k,y_mpc(1,:), 'Color', [0.38 .38 0.38], 'LineWidth', 2); grid on; hold on
plot(k,y_mpc(2,:), '--', 'Color', [0.38 .38 0.38], 'LineWidth', 2)
plot(k, y_mpcL(1,:), 'k', 'LineWidth', 2)
plot(k, y_mpcL(2,:), 'k--', 'LineWidth', 2)
hline           = refline([0 rs(1)]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline           = refline([0 rs(2)]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'Position of the Cart Classical MPC','Angle of the Pendulum Classical MPC', ...
'Position of the Cart Laguerre MPC', 'Angle of the Pendulum Laguerre MPC'}, ...
        'Location','east')
title('Comparison Classic MPC Design and MPC Design with Laguerre functions')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
supersizeme(f3, 1.5)

xNN    = zeros(4,1); u_mpcL = [0,u_mpcL];
    for i = 1:Nsim
        [yNN, xNN] = simRealResponse(u_mpcL(i), xNN);
        xN(:,i)    = xNN; yN(:,i)    = yNN;
    end

f4 = figure;
subplot(211)
plot(k, yN(1,:), 'Color', [0 .6 1], 'LineWidth', 2); grid on; hold on
plot(k, yN(2,:), 'c', 'LineWidth', 2)
```

```
hline           = refline([0 rs(1)]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
legend({'Position of the Cart','Angle of the Pendulum'},'Location', ...
        'northwest'); title('The True Process Response')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
subplot(212)
plot(k, yN(1,:), 'Color', [0 .6 1], 'LineWidth', 2.5); grid on; hold on
plot(k, yN(2,:), 'c', 'LineWidth', 2.5)
plot(k,y_mpcL(1,:), 'k', 'LineWidth', 2)
plot(k,y_mpcL(2,:), 'k--', 'LineWidth', 2)
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':'; axis([0 200 -20 20])
title('Noise-Free Process Response compared to the True Process Response')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
legend({'True Position of the Cart','True Angle of the Pendulum', ...
        'Estimated Position of the Cart', 'Estimated Angle of the Pendulum'},'Location', ...
        'southeast')
supersizeme(f4, 1.5)

%% Save useful data in data construct
% save variable in the data.mat file
u_mpc = u_mpcL;
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sysdata = load('data.mat');

%% The Free Energy principle
% Model Predictive Control with Real-Time State estimation such that the
% output simulation does not blow-up,
StateEst = 1; % State estimation according to Free Energy principle on
Laguerre = 1;

% Determine set-point, or reference signal:
% Reference (Position of cart and pendulum [m] and [rad]
rs = [10;0]; Rs = [rs(1)*ones(Nsim,1),rs(2)*ones(Nsim,1)];
ru = 1; Ru = diag(ru * ones(1,Nc));

[u_mpc, yNNN, xNNN] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, StateEst, ...
                      Laguerre); % Returns True Process response

% Save useful data in data construct, save variable in the data.mat file
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sysdata = load('data.mat');

k = 1:Nsim;
f5 = figure;
plot(k, u_mpc,'k', 'LineWidth', 2); grid on; hold on
plot(k, u_mpcL(1:Nsim), 'Color', [0.38 .38 0.38], 'LineWidth', 2)
title('The generated Control Trajectory by the MPC Design')
xlabel('Sampling Instant'); ylabel('Control input in Voltage [V]')
legend({'MPC Design with FE state estimation','MPC Design with simple state estimation'}, ...
       'Location', 'southeast' )
supersizeme(f5, 1.5)

f6 = figure;
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 2); grid on; hold on
plot(k, yNNN(2,:), 'c', 'LineWidth', 2)
plot(k,y_mpcL(1,:), 'k', 'LineWidth', 2)
plot(k,y_mpcL(2,:), 'k--', 'LineWidth', 2)
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
```

```
hline.LineWidth = 1.5; hline.LineStyle = ':';
title('Noise-Free Process Response compared to the True Process Response')
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
legend({'True Position of the Cart','True Angle of the Pendulum', ...
        'Estimated Position of the Cart', 'Estimated Angle of the Pendulum'}, ...
        'Location', 'east')
supersizeme(f6, 1.5)

% Set the initial conditions:
x0 = zeros(n,1); P0 = 0.1*eye(n);
[mu_x, Sigma_x, Sigma_x_bar] = FreeEnergyStateEst(sysdata, yNNN, x0, P0);

% State Estimation that is done after the control trajectory is generated:
% Is still different than the one inside the mpc control script, more noise
% on the output.

f7 = figure;
subplot(2,2,1)
plot(k,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3); grid on; hold on
plot(k, mu_x(:,1), 'k', 'LineWidth', 2)
title('Estimation of \mu_{x1} according to the FEP')
xlabel('Sampling Instant'); ylabel('Position of the Cart [m]')
legend({'The True State: x','The estimated State: \mu_x'}, ...
        'Location', 'southeast')
subplot(2,2,2)
plot(k,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3); grid on; hold on
plot(k, mu_x(:,2), 'k', 'LineWidth', 2)
xlabel('Sampling Instant'); ylabel('Velocity of the Cart [m/s]')
title('Estimation of \mu_{x2} according to the FEP')
subplot(2,2,3)
plot(k,xNNN(3,:), 'c', 'LineWidth', 2.5); grid on; hold on
plot(k, mu_x(:,3), 'k--', 'LineWidth', 2)
xlabel('Sampling Instant'); ylabel('Angle of the Pendulum [rad]')
title('Estimation of \mu_{x3} according to the FEP')
legend({'The True State: x','The estimated State: \mu_x'}, ...
        'Location', 'northeast')
subplot(2,2,4)
plot(k,xNNN(4,:),  'c', 'LineWidth', 2.5); grid on; hold on
plot(k, mu_x(:,4), 'k--','LineWidth', 2)
xlabel('Sampling Instant'); ylabel('Angular velocity of the Pendulum [rad/s]')
title('Estimation of \mu_{x4} according to the FEP')
supersizeme(f7, 1.5)

% Learning the Model matrices according to the Free Energy principle:
% Initial conditions, the hyper parameters are neglected
Ealpha = 0; Ebeta = 0; mu_x = mu_x.';
[muA, SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst(sysdata, yNNN, ...
                      mu_x, Sigma_x, Sigma_x_bar, Ealpha, Ebeta);

% Predicion of Process Response with every Learned model, for every
% sampling instant there is a corresponding A and B matrix:
uSim = [0, u_mpc];

for ii = 1:Nsim
    yyy  = zeros(m,1);
        for jj = 1:Nsim
            yest(:,jj) = yyy;
            Xnew = muA(:,:,ii)*(mu_x(:,jj)) + muB(:,:,ii)*uSim(jj);
            yyy = Cd*Xnew;
        end
    yhat(:,:,ii) = yest;
```

```
end

f8 = figure;
subplot(121)
    for ii = 15:Nsim
        if ii == Nsim
             yyyy = yhat(:,:,ii); h2 = plot(k, yyyy(1,:), 'k');
        else
            yyyy = yhat(:,:,ii); plot(k, yyyy(1,:), 'k')
        end
    hold on
    end
h1 = plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 2.5);
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':'; ylim([-5 15])
grid on; xlabel('Sampling Instant'); ylabel('Position of the cart in [m]')
legend([h1, h2],{'True Position of the Cart', 'Estimates of Position of the Cart'})
title({'\textbf{\hat y _1 simulated with every:} (\mu_A)_{1:K}, (\mu_B)_{1:K}'},'Interpreter',
subplot(122)
    for ii = 15:Nsim
        if ii == Nsim
             yyyy = yhat(:,:,ii); h2 = plot(k, yyyy(2,:), 'k');
        else
            yyyy = yhat(:,:,ii); plot(k, yyyy(2,:), 'k')
        end
        hold on
    end
h1 = plot(k, yNNN(2,:), 'Color', 'c', 'LineWidth', 2);
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':'; ylim([-5 15]);
grid on; xlabel('Sampling Instant'); ylabel('Angle of the Pendulum in[rad]')
legend([h1, h2],{'True Angle of the Pendulum', 'Estimates of Angle of the Pendulum'})
title({'\textbf{\hat y _2 simulated with every:} (\mu_A)_{1:K}, (\mu_B)_{1:K}'},'Interpreter',
supersizeme(f8, 1.5)

%%%%%%%%%%%%%%%%%%%%VAF%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for ll = 1:Nsim
    yvaf = yhat(:,:,ll);
    vaf1(ll) = (1 - var(yNNN(1,:)-yvaf(1,:))/var(yNNN(1,:)))*100;
    vaf2(ll) = (1 - var(yNNN(2,:)-yvaf(2,:))/var(yNNN(2,:)))*100;
end
yhatbest = yhat(:,:,end);

f9 = figure;
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 2.5); hold on; grid on
plot(k, yNNN(2,:), 'Color', 'c', 'LineWidth', 2.5)
plot(k, yhatbest(1,:), 'k:', 'LineWidth', 2)
plot(k, yhatbest(2,:), 'k--', 'LineWidth', 2)
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
title({'\textbf{The Estimated Process Response simulated with} A_e, B_e: \hat y'},'Interpreter
xlabel('Samping Instant'); ylabel('Process Response in [m] and [rad]')
legend({'True Position of the Cart', 'True Angle of the Pendulum' , ...
       'Estimated Position of the Cart', 'Estimated Angle of the Pendulum', ...
        },'Location', 'east')
supersizeme(f9, 1.5)

f10 = figure;
subplot(121)
```

```
bar(k, vaf1, 'k'); grid on; ylim([0 100]); xlim([0 100])
title('The Variance Accounted For; Position of the Cart')
xlabel('Sampling instant'); ylabel('VAF in [%]')
subplot(122)
bar(k, vaf2, 'k'); grid on; ylim([0 100]); xlim([0 100])
title('The Variance Accounted For; Angle of the Pendulum')
xlabel('Sampling instant'); ylabel('VAF in [%]')
supersizeme(f10, 1.5)

%% The Free Energy Calculations
% Calculates the Free Energy with respect to A and B for the state-space
% representation derived from first principles and state sequence that
% should resemble the actual system, can be regarded as the True Free
% Energy:

AforFE = zeros(n); BforFE = zeros(n,1); Pp =0.1*eye(n);
for i=1:Nsim
    Sigmaxp(:,:,i) = Ad*Pp*(Ad.') + Q.*eye(n);
    K  = (Sigmaxp(:,:,i)*(Cd.')) / (Cd*Sigmaxp(:,:,i)*(Cd.')+R*eye(m));
    Pp = (eye(n)-K*Cd)*Sigmaxp(:,:,i);

    AforFE(:,:,i) = Ad; BforFE(:,:,i) = Bd;
end

[FAtrue, FBtrue]   = FreeEnergy(AforFE, 0, BforFE, 0, ...
                                xNNN, Sigmaxp, u_mpc, Nsim);

f11 = figure;
subplot(121); grid on
plot(k,FAtrue, 'k', 'LineWidth', 1.5)
xlabel('Sampling Instant'); ylabel('The Free Energy');
title('The True Free Energy with respect to A')
subplot(122); grid on
plot(k,FBtrue, 'k', 'LineWidth', 1.5)
title('The True Free Energy with respect to B')
xlabel('Sampling Instant'); ylabel('The Free Energy')
supersizeme(f11, 1.5)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Some Lines to check the Free Energy:
AFreeTestpart = [0.001, 0.01, 0.1, 0.5, 1]; randomA = randn(n);

for run = 1:6
 if run == 6
     for i=1:Nsim
        AforFE(:,:,i) = Ad; BforFE(:,:,i) = Bd;
     end
 else
     for i=1:Nsim
        AforFE(:,:,i) = AFreeTestpart(run)*randomA;
        BforFE(:,:,i) = Bd;
     end
 end

[FAtruetest, FBtruetest]   = FreeEnergy(AforFE, 0, ...
                         BforFE, 0, xNNN, Sigmaxp, u_mpc, Nsim);

 FAtruetest250(run,:) = FAtruetest(250);
 FBtruetest250(run,:) = FBtruetest(250);
end

% Estimation of Free Energy with the estimated model matrices:
```

```
[Fa2, Fb2]    = FreeEnergy(muA, SigmaA, muB, SigmaB, ...
                           mu_x, Sigma_x_bar, u_mpc, Nsim);

f12 = figure;
subplot(121)
plot(k, FAtrue, 'c', 'LineWidth', 3); hold on; grid on
plot(k, Fa2, 'k', 'LineWidth', 2)
title('The Free Energy with respect to A')
xlabel('Sampling Instant'); ylabel('The Free Energy');
legend('The True Free Energy','The estimated Free Energy', 'Location', 'southeast')
subplot(122)
plot(k, FBtrue, 'c', 'LineWidth', 3); hold on; grid on
plot(k, Fb2, 'k', 'LineWidth', 2)
title('The Free Energy with respect to B')
xlabel('Sampling Instant'); ylabel('The Free Energy');
legend('The True Free Energy','The estimated Free Energy', 'Location', 'southeast')
supersizeme(f12, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Subspace Identification
% For comparison with the Free Energy Learning algorithm, although it
% is currently not a very fair comparison
s = 8;   % Twice the Process order
N = Nsim;

% Construct Hankel matrices
UsN = u_mpc(:,1:N-s+1); YsN =  yNNN(:,1:N-s+1);
for i=2:s
    UsN = [UsN; u_mpc(:,i:N-s+i)]; YsN = [YsN;  yNNN(:,i:N-s+i)];
end

PI      = eye(size(UsN,2),s) - (UsN.')*(UsN*(UsN.').^-1);
x       = YsN*PI; [U,S,V] = svd(x);

% compute RQ factorization
r=triu(qr([UsN' YsN']))'; R22 = r(s*p+1:s*(p+m),s*p+1:s*(p+m));

% compute svd
[uu,Ss,vv]=svd(R22); %figure,semilogy(diag(ss),'xr','LineWidth',2,'MarkerSize',12),
n = 4;
Ct = uu(1:m,1:n); Ct1 = U(1:m,1:n);

% compute least squares
At = uu(1:m*(s-1),1:n)\uu(m+1:s*m,1:n);
At1 = U(1:m*(s-1),1:n)\U(m+1:s*m,1:n);

[B1,D1] = dac2bd(At1,Cd, u_mpc.', yNNN.');

for jj = k
    Xnew = At1*(mu_x(:,jj)) + B1*u_mpc(jj);
    yyy = Cd*Xnew; yhatSubspace(:,jj) = yyy;
end

vafsubspace1 = (1 - var(yNNN(1,:)-yhatSubspace(1,:))/var(yNNN(1,:)))*100;
vafsubspace2 = (1 - var(yNNN(2,:)-yhatSubspace(2,:))/var(yNNN(2,:)))*100;

f12 = figure;
subplot(121)
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3); hold on; grid on
plot(k,yhatbest(1,:), 'k', 'LineWidth', 2)
plot(k,yhatSubspace(1,:), 'k:', 'LineWidth', 2)
```

```
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Samping Instant'); ylabel('Position of the Cart in [m] ')
title('Comparison True and Estimated Process Response: y_1')
legend({ 'True Position of the Cart', 'Position of the Cart Free Energy Estimation', ...
       'Position of the Cart Subspace Estimation'},'Location', 'southeast')
subplot(122)
plot(k, yNNN(2,:), 'Color', 'c', 'LineWidth', 3); hold on; grid on
plot(k,yhatbest(2,:), 'k', 'LineWidth', 2)
plot(k,yhatSubspace(2,:), 'k:', 'LineWidth', 2)
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Samping Instant'); ylabel('Angle of the Pendulum in [rad]')
title('Comparison True and Estimated Process Response: y_2')
legend({'True Angle Position of the Pendulum', 'Angle of the Pendulum Free Energy Estimation',
        'Angle of the Pendulum Subspace Estimation'},'Location', 'southeast')
supersizeme(f12, 1.5)

% To improve the estimate, incorporation of the Colored Noise
N = Nsim - 2*s + 1; L = [];

Up = zeros(N,p * s); Uf = zeros(N,p * s);
Yf = zeros(N,m * s); Yp = zeros(N,m * s);

  for k = (1:s)
    Up(:,(k-1) * p+1:k * p) = (u_mpc(:,k:N+k-1).');
    Uf(:,(k-1) * p+1:k * p) = (u_mpc(:,s+k:N+s+k-1).');
    Yp(:,(k-1) * m+1:k * m) = (yNNN(:,k:N+k-1).');
    Yf(:,(k-1) * m+1:k * m) = (yNNN(:,s+k:N+s+k-1).');
  end

L = triu(qr([Uf Up Yp Yf])).';
L = L(1:2 * (p+m) * i,1:2 * (p+m) * s);
Lb = L((2 * p+m) * s+1:2 * (p+m) * s, p * s+1:(2 * p+m) * s); %%R32

[Un,Sn,Vn] = svd(Lb);

At11 = Un(1:m*(s-1),1:n)\Un(m+1:s*m,1:n)
[B11,D11] = dac2bd(At11,Cd, u_mpc.', yNNN.')

for jj = 1:Nsim
     Xnew = At11*(mu_x(:,jj)) + B11*u_mpc(jj);
     yyy = Cd*Xnew; yhatSubspace1(:,jj) = yyy;
end

vafsubspace11 = (1 - var(yNNN(1,:)-yhatSubspace1(1,:))/var(yNNN(1,:)))*100;
vafsubspace21 = (1 - var(yNNN(2,:)-yhatSubspace1(2,:))/var(yNNN(2,:)))*100;

k = 1:Nsim;

f13 = figure;
subplot(121)
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3); hold on; grid on
plot(k,yhatbest(1,:), 'k', 'LineWidth', 2)
plot(k,yhatSubspace1(1,:), 'k:', 'LineWidth', 2)
hline  = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Samping Instant'); ylabel('Position of the Cart in [m] ')
title('Comparison True and Estimated Process Response: y_1')
legend({ 'True Position of the Cart', 'Position of the Cart Free Energy Estimation', ...
       'Position of the Cart Subspace Estimation'},'Location', 'southeast')
```

```
subplot(122)
plot(k, yNNN(2,:), 'Color', 'c', 'LineWidth', 3); hold on; grid on
plot(k,yhatbest(2,:), 'k', 'LineWidth', 2)
plot(k,yhatSubspace1(2,:), 'k:', 'LineWidth', 2)
hline = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
xlabel('Samping Instant'); ylabel('Angle of the Pendulum in [rad]')
title('Comparison True and Estimated Process Response: y_2')
legend({'True Angle Position of the Pendulum', 'Angle of the Pendulum Free Energy Estimation',
        'Angle of the Pendulum Subspace Estimation'},'Location', 'southeast')
supersizeme(f13, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %% State Estimation and Learning inside the MPC Design
% % Up till now the learning is done outside of the MPC design, this
% % corresponds to batch Learning oppose to simultaneous Learning. The
% % following lines of script considers simultaneos Learning more suitable
% % for Real-Time Applications. Second Model-Control algorithm.
%
% Set Predictive and Control Horizon:

% Initial model matrices, ss(A,B,C,D), Discrete-Time:
Ampc = Ad + 0.01*randn(4); %Ampc = randn(n);
Bmpc = Bd + 0.01*randn(4,1); %Bmpc = randn(n,1);
Cmpc = [ 1, 0, 0, 0; 0, 0, 1, 0]; Dmpc = [0;0];

SysMPC = ss(Ampc, Bmpc, Cmpc, Dmpc, 0.1);

[u_Learning, y_Learning, x_Learning, Atable, Btable]  ...
                          =FreeEnergyAlgorithm(SysMPC, Nsim, u_mpc, yNNN);

f14 = figure;
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 2.5); hold on; grid on
plot(k, yNNN(2,:), 'Color', 'c', 'LineWidth', 2.5)
plot(k, y_Learning(1,2:Nsim+1), 'k:', 'LineWidth', 2)
plot(k, y_Learning(2,2:Nsim+1), 'k--', 'LineWidth', 2)
hline = refline([0 rs(1)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline  = refline([0 rs(2)]); hline.Color = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
title({'\textbf{The Estimated Process Response simulated with} A_e, B_e: \hat y \textbf{by the
xlabel('Samping Instant')
ylabel('Process Response in [m] and [rad]')
legend({'True Position of the Cart', 'True Angle of the Pendulum' , ...
       'Estimated Position of the Cart', 'Estimated Angle of the Pendulum', ...
        },'Location', 'east')
supersizeme(f14, 1.5)


f15 = figure;
subplot(2,2,1)
plot(1:Nsim,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3); grid on; hold on
plot(1:Nsim, x_Learning(1,:), 'k', 'LineWidth', 2)
xlabel('Sampling Instant'); ylabel('Position of the Cart [m]')
title('Estimation of \mu_{x1} according to the FEP')
legend({'The True State: x','The estimated State: \mu_x'}, ...
        'Location', 'southeast')
subplot(2,2,2)
plot(1:Nsim,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3); grid on; hold on
plot(1:Nsim, x_Learning(2,:), 'k', 'LineWidth', 2)
xlabel('Sampling Instant');  ylabel('Velocity of the Cart [m/s]')
```

```
title('Estimation of \mu_{x2} according to the FEP')
subplot(2,2,3)
plot(1:Nsim,xNNN(3,:), 'c', 'LineWidth', 2.5); grid on; hold on
plot(1:Nsim, x_Learning(3,:), 'k--', 'LineWidth', 2)
xlabel('Sampling Instant'); ylabel('Angle of the Pendulum [rad]')
title('Estimation of \mu_{x3} according to the FEP')
legend({'The True State: x','The estimated State: \mu_x'}, ...
        'Location', 'northeast')
subplot(2,2,4)
plot(1:Nsim,xNNN(4,:),  'c', 'LineWidth', 2.5); grid on; hold on
xlabel('Sampling Instant'); ylabel('Angular velocity of the Pendulum [rad/s]')
 plot(1:Nsim, x_Learning(4,:), 'k--','LineWidth', 2)
title('Estimation of \mu_{x4} according to the FEP')
supersizeme(f15, 1.5)

Arandom = [0, 0.001, 0.01, 0.05, 0.1, 1];
Brandom = [0, 0.001, 0.01, 0.05, 0.1, 1];

for run = 1:6
Ampc = Ad + Arandom(run)*randn(4); Bmpc = Bd + Arandom(run)*randn(4,1);
Cmpc = [ 1, 0, 0, 0; 0, 0, 1, 0]; Dmpc = [0;0];

SysMPC = ss(Ampc, Bmpc, Cmpc, Dmpc, 0.1);

Ainitial(:,:,run) = Ampc; Binitial(:,:,run) = Bmpc;

[u_Learning, y_Learning, x_Learning, Atable, Btable]  ...
                           =FreeEnergyAlgorithm(SysMPC, Nsim, u_mpc, yNNN);

xtest(:,:,run) = x_Learning; Atest(:,:,run) = Atable(:,:,end);
Btest(:,:,run) = Btable(:,:,end);
end

f16 = figure;
for run = 2:5
x = xtest(:,:,run);
   if run == 2
   subplot(4,4,1); plot(k,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_1 in [m]'); ylim([0 15])
   hold on; plot(k, x(1,:), 'k', 'LineWidth', 2); title('0.001 Randomness')
   subplot(4,4,5); plot(k,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_2 in [m/s]')
   hold on; plot(k, x(2,:), 'k', 'LineWidth', 2)
   subplot(4,4,9); plot(k,xNNN(3,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_3 in [rad]')
   hold on; plot(k, x(3,:), 'k', 'LineWidth', 2)
   subplot(4,4,13); plot(k,xNNN(4,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_4 [rad/s]'); ylim([-20 20])
   hold on; plot(k, x(4,:), 'k', 'LineWidth', 2)
   elseif run == 3
   subplot(4,4,2); plot(k,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_1 in [m]'); ylim([0 15])
   hold on; plot(k, x(1,:), 'k', 'LineWidth', 2); title('0.01 Randomness')
   subplot(4,4,6); plot(k,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_2 in [m/s]')
   hold on; plot(k, x(2,:), 'k', 'LineWidth', 2)
   subplot(4,4,10); plot(k,xNNN(3,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_3 in [rad]')
   hold on; plot(k, x(3,:), 'k', 'LineWidth', 2)
   subplot(4,4,14); plot(k,xNNN(4,:), 'Color', [0 .6 1], 'LineWidth', 3)
   grid on; xlabel('Sampling Instant'); ylabel('x_4 in [rad/s]'); ylim([-20 20])
```

```
    hold on; plot(k, x(4,:), 'k', 'LineWidth', 2)
    elseif run ==4
    subplot(4,4,3); plot(k,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_1 in [m]'); ylim([0 15])
    hold on; plot(k, x(1,:), 'k', 'LineWidth', 2); title('0.05 Randomness')
    subplot(4,4,7); plot(k,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_2 in [m/s]')
    hold on; plot(k, x(2,:), 'k', 'LineWidth', 2)
    subplot(4,4,11); plot(k,xNNN(3,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_3 in [rad]')
    hold on; plot(k, x(3,:), 'k', 'LineWidth', 2)
    subplot(4,4,15); plot(k,xNNN(4,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_4 in [rad/s]'); ylim([-20 20])
    hold on; plot(k, x(4,:), 'k', 'LineWidth', 2)
    else
    subplot(4,4,4); plot(k,xNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_1 in [m]'); ylim([0 15])
    hold on; plot(k, x(1,:), 'k', 'LineWidth', 2); title('0.1 Randomness')
    subplot(4,4,8); plot(k,xNNN(2,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_2 in [m/s]')
    hold on; plot(k, x(2,:), 'k', 'LineWidth', 2)
    subplot(4,4,12); plot(k,xNNN(3,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_3 in [rad]')
    hold on; plot(k, x(3,:), 'k', 'LineWidth', 2)
    subplot(4,4,16); plot(k,xNNN(4,:), 'Color', [0 .6 1], 'LineWidth', 3)
    grid on; xlabel('Sampling Instant'); ylabel('x_4 in [rad/s]'); ylim([-20 20])
    hold on; plot(k, x(4,:), 'k', 'LineWidth', 2)
    end
end

supersizeme(f16, 1.5)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CheckVAF

for run = 1:6
% Set the initial conditions:
x0 = zeros(n,1); P0 = 0.1*eye(n);

[u_mpc, yNNN, xNNN] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, ...
       StateEst, Laguerre);  % Returns True Process response

sysdata.u_mpc = u_mpc;

[mu_x, Sigma_x, Sigma_x_bar] = FreeEnergyStateEst(sysdata, yNNN, x0, P0);

mu_x = mu_x.';
[muA, SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst(sysdata, yNNN, ...
                      mu_x, Sigma_x, Sigma_x_bar, Ealpha, Ebeta);

uSim = [0, u_mpc];
        for jj = 1:Nsim
            Xnew = muA(:,:,end)*(mu_x(:,jj)) + muB(:,:,end)*uSim(jj);
            yhatcheck(:,jj) = Cd*Xnew;
        end
checkvaf1(run) = (1 - var(yNNN(1,:)-yhatcheck(1,:))/var(yNNN(1,:)))*100;
checkvaf2(run) = (1 - var(yNNN(2,:)-yhatcheck(2,:))/var(yNNN(2,:)))*100;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CheckNoise
```

```
NoiseTest = [0.1, 0.05 ,0.01];

for run = 1:3
% Determine set-point, or reference signal:
% Reference (Position of cart and pendulum [m] and [rad]
rs = [10;0]; Rs = [rs(1)*ones(Nsim,1),rs(2)*ones(Nsim,1)];
ru = 1; Ru = diag(ru * ones(1,Nc));

Q = NoiseTest(run); R = NoiseTest(run);
% Save useful data in data construct, save variable in the data.mat file
save('data.mat','Ad', 'Bd', 'Cd', 'Dd','Q', 'R', 'u_mpc')
sysdata = load('data.mat');

[u_mpc, yNNN, xNNN] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, ...
    StateEst, Laguerre); % Returns True Process response

[mu_x, Sigma_x, Sigma_x_bar] = FreeEnergyStateEst(sysdata, yNNN, x0, P0);

mu_x = mu_x.';
[muA, SigmaA, muB, SigmaB] = FreeEnergyModelParametersEst(sysdata, yNNN, ...
                        mu_x, Sigma_x, Sigma_x_bar, Ealpha, Ebeta);

    for jj = 1:Nsim
            Xnew = muA(:,:,end)*(mu_x(:,jj)) + muB(:,:,end)*u_mpc(jj);
            yhatcheck(:,jj) = Cd*Xnew;
    end

if run == 1; f17 = figure; end

subplot(1,3,run); grid on;hold on
plot(k, yNNN(1,:), 'Color', [0 .6 1], 'LineWidth', 3)
plot(k, yNNN(2,:), 'c', 'LineWidth', 3)
plot(k,yhatcheck(1,:), 'k', 'LineWidth', 2)
plot(k,yhatcheck(2,:), 'k--', 'LineWidth', 2)
hline           = refline([0 rs(1)]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
hline           = refline([0 rs(2)]); hline.Color     = [0.5 0.5 0.5];
hline.LineWidth = 1.5; hline.LineStyle = ':';
title(['Noise Level=' num2str(Q)]); ylim([-5 15])
xlabel('Sampling Instant'); ylabel('Process Response in [m] and [rad]')
legend({'y_1','y_2', '\hat{y}_1', '\hat{y}_2'},  ...
        'Interpreter','Latex','Location', 'east')
end
supersizeme(f17, 1.5)
```

```
function [FA, FB] = FreeEnergy(muA, SigmaA, muB, SigmaB, mux, ...
                                Sigmaxp, u, Nsim)
% Calculates the Free Energy
if SigmaA ==0
    for k = 2:Nsim
        Sxp = 1\Sigmaxp(:,:,k-1);
        vecmuApart = muA(:,:,k-1); vecmuA = vecmuApart(:);
        vecmuATvecmuA = vecmuA*(vecmuA.');
        trvecmuATvecmuA = mat2cell(vecmuATvecmuA,[4 4 4 4], [4 4 4 4]);
        trtrvecmuATvecmuASxp = trace(trvecmuATvecmuA{1,1}*Sxp);
        Fa = -log(1/sqrt(2*pi*det(Sigmaxp(:,:,k-1)))) ...
             -(1/2)*(mux(:,k).')*Sxp*muA(:,:,k-1)*mux(:,k-1) ...
             -(1/2)*(mux(:,k-1).')* (muA(:,:,k-1).')*Sxp*mux(:,k) ...
             +(1/2)*(mux(:,k-1).')* (muA(:,:,k-1).')*Sxp*muB(:,:,k-1)*u(k-1) ...
```

```
            +(1/2)*(mux(:,k-1).')* trtrvecmuATvecmuASxp * mux(:,k-1) ...
            +(1/2)*u(k-1)*(muB(:,:,k-1).')*Sxp*muA(:,:,k-1)*mux(:,k-1);
        FA(k) = Fa;

        muBpart = muB(:,:,k-1); muBTmuB = muBpart*(muBpart.');
        trmuBTmuBSxp = trace(muBTmuB*Sxp);
        Fb = -(1/2)*log(1/sqrt(2*pi*det(Sigmaxp(:,:,k-1)))) ...
            -(1/2)*(mux(:,k).')*Sxp*muB(:,:,k-1)*u(k-1) ...
            +(1/2)*(mux(:,k-1).')*(muA(:,:,k-1).')*Sxp* muB(:,:,k-1)*u(k-1) ...
            -(1/2)*u(k-1)* (muB(:,:,k-1).')*Sxp* mux(:,k) ...
            +(1/2)*u(k-1)* (muB(:,:,k-1).')*Sxp* muA(:,:,k-1)*mux(:,k-1) ...
            +(1/2)*u(k-1)* trmuBTmuBSxp * u(k-1);
        FB(k) = Fb;
    end
else

    for k = 2:Nsim
        Sxp = 1\Sigmaxp(:,:,k-1);
        vecmuApart = muA(:,:,k-1); vecmuA = vecmuApart(:);
        vecmuATvecmuA =  vecmuA*(vecmuA.');
        trvecmuATvecmuA = mat2cell(vecmuATvecmuA,[4 4 4 4], [4 4 4 4]);
        trtrvecmuATvecmuASxp = trace(trvecmuATvecmuA{1,1}*Sxp);
        Fa = -log(1/sqrt(2*pi*det(Sigmaxp(:,:,k-1)))) ...
            -(1/2)*(mux(:,k).')*Sxp*muA(:,:,k-1)*mux(:,k-1) ...
            -(1/2)*(mux(:,k-1).')* (muA(:,:,k-1).')*Sxp*mux(:,k) ...
            +(1/2)*(mux(:,k-1).')* (muA(:,:,k-1).')*Sxp*muB(:,:,k-1)*u(k-1) ...
            +(1/2)*u(k-1)*(muB(:,:,k-1).')*Sxp*muA(:,:,k-1)*mux(:,k-1) ...
            +(1/2)*(mux(:,k-1).')* trtrvecmuATvecmuASxp * mux(:,k-1);
            -(1/2)*log(det(SigmaA(:,:,k-1)));
        FA(k) = Fa;

        muBpart = muB(:,:,k-1); muBTmuB =  muBpart*(muBpart.');
        trmuBTmuBSxp = trace(muBTmuB*Sxp);
        Fb = -(1/2)*log(1/sqrt(2*pi*det(Sigmaxp(:,:,k-1)))) ...
            -(1/2)*(mux(:,k).')*Sxp*muB(:,:,k-1)*u(k-1) ...
            +(1/2)*(mux(:,k-1).')*(muA(:,:,k-1).')*Sxp* muB(:,:,k-1)*u(k-1) ...
            -(1/2)*u(k-1)* (muB(:,:,k-1).')*Sxp* mux(:,k) ...
            +(1/2)*u(k-1)* (muB(:,:,k-1).')*Sxp* muA(:,:,k-1)*mux(:,k-1) ...
            +(1/2)*u(k-1)* trmuBTmuBSxp * u(k-1);
            -(1/2)*log(det(SigmaB(:,:,k-1)));
        FB(k) = Fb;
    end
end
```

```
function [u_mpc, y_mpc, x_mpc, Atable, Btable] = FreeEnergyAlgorithm(Sys, Nsim, u, yNNN)
% Set Initial input and output:
u = [0, u]; % u0

% Initial data-store matrices:
y_mpc = zeros(2,Nsim); u_mpc = zeros(4,Nsim);

% State estimation from the Free Energy principle is provided:
% Initial conditions:
P0 = 0.1*ones(4); x0 = [0;0;0;0];
sumx0x0T = zeros(4); sumx0x1T = zeros(4); sumx0u1BT = zeros(4);
sumu0u0T = 0; sumu0x1T = [0,0,0,0]; sumu0x1AT = [0,0,0,0];

for kk=1:Nsim
% Calculates the next state estimation:
```

```
[x1, P1] = MPCFreeEnergyStateEst(Sys, yNNN(:,kk), u(kk), x0, P0);

[muA, muB, sumx0x0T, sumx0x1T, sumx0u1BT, sumu0u0T, sumu0x1T, sumu0x1AT]...
= FreeEnergyModelParametersEstMPC(Sys, x0, x1, u(kk), sumx0x0T, sumx0x1T, ...
                                  sumx0u1BT, sumu0u0T, sumu0x1T, sumu0x1AT);

SysMPC.A = muA; SysMPC.B = muB; Atable(:,:,kk) = muA; Btable(:,:,kk) = muB;

y_mpc(:,kk+1) = [1,0,0,0; 0,0,1,0]*(muA*x1 + muB*u(kk+1));

% Set the current values equivalent to past values for the next iteration:
x0 = x1; P0 = P1; x_mpc(:,kk) = x0;
end
end
```

A.4. JOINT MATLAB SCRIPTS

A.4.1. JOINT MPC DESIGN SCRIPTS

```
function [SysA] = AugmentedSS(SysD)
% Used to convert the provided Discrete-Time system into a system augmented
% with an integrator, subscript of the augmented state-space representation
% is a
Ad = SysD.A; Bd = SysD.B; Cd = SysD.C;

[m,n]=size(Cd); [n,p]=size(Bd);

Aa=eye(n+m,n+m); Aa(1:n,1:n)=Ad; Aa(n+1:n+m,1:n)=Cd*Ad;
Ba=zeros(n+m,p); Ba(1:n,:)=Bd; Ba(n+1:n+m,:)=Cd*Bd;
Ca=zeros(m,n+m); Ca(:,n+1:n+m)=eye(m,m);
Da = [];

SysA = ss(Aa, Ba, Ca, Da);
end
```

```
function [PhiPhi, PhiF, PhiR] = MPCmatrices(SysA, Np, Nc)
Aa = SysA.A; Ba = SysA.B; Ca = SysA.C; Da = SysA.D;
sys = load('data.mat'); Ad = sys.Ad; Cd = sys.Cd;
m = size(Cd,1); % m is the number of outputs, n is the number of states
n = size(Ad,1); % p is the number of inputs
h(1:m,:) = Ca; F(1:m,:) = Ca*Aa;
if n == 1
    for kk=2:Np; h(kk,:)=h(kk-1,:)*Aa; F(kk,:)= F(kk-1,:)*Aa; end
else
    for kk = 2 : m : m*(Np-1)
    h(kk+1:kk+m,:) = h(kk-m+1:kk,:)*Aa; F(kk+1:kk+m,:) = F(kk-m+1:kk,:)*Aa;
    end
end
xx = h*Ba; Phi = zeros(m*Np,Nc); %declare the dimension of Phi
Phi(:,1) = xx; % first column of Phi
for ll = 2:Nc
Phi(:,ll) = [zeros(m*(ll-1),1); xx(1:m*(Np-ll+1),1)]; %Toeplitz matrix
end
% Useful Calculations for the control Law:
BarRs  = ones(Np*m,2); PhiPhi = Phi.'*Phi; PhiF = Phi.'*F;
PhiR   = Phi.'*BarRs;
end
```

```
function [A,L0]=lagd(a,N)
v(1,1)=a; L0(1,1)=1;

for k=2:N
    v(k,1)=(-a).^(k-2)*(1-a*a);
    L0(k,1)=(-a).^(k-1);
end

L0=sqrt((1-a*a))*L0; %L(0)^T = sqrt(\beta)[1, -a, a^2, -a^3, ..
A(:,1)=v;

for i=2:N
    A(:,i)=[zeros(i-1,1);v(1:N-i+1,1)];
end
```

A.4.2. JOINT FREE ENERGY SCRIPTS

```
function [mu_x, Sigma_x, Sigma_x_bar] = FreeEnergyStateEst(sys, y, x0, P0)
% Simulation time:
N = size(y,2);

% Retrieve System model matrices
A = sys.Ad; B = sys.Bd; C = sys.Cd; D = sys.Dd;
Q = sys.Q;  R = sys.R;
u = [0,sys.u_mpc];

% Model order:
n = size(A,1); m = size(C,1);

mu_x_prior = x0; P_prior = P0; Q = Q*eye(n); R = R*eye(m);

for i = 1:N
    %%%% Prediction step
    % mu_x_bar = Am * mu_x_prior; % Autonomous System
    mu_x_bar = A * mu_x_prior + B*u(i); % Prediction State
    P_bar = A*P_prior*(A.') + Q;        % Prediction Variance
    Sigma_x_bar(:,:,i) = P_bar;
    %%%% Measurement step
    % Calculation of Kalman Gain
    K = P_bar*(C.')*inv(C*P_bar*(C.')+R);
    % Update of state and variance (Correction with current measurement)
    mu_x_post = mu_x_bar + K*(y(:,i)-C*mu_x_bar);
    P_post = (eye(n) - K*C)*(P_bar);
    % Store Information in data constructs
    mu_x(i,:)=mu_x_post;
    Sigma_x(:,:,i) = P_post;
    % Posterior becomes prior
    mu_x_prior = mu_x_post;
    P_prior = P_post;
end
```

```
function [u_mpc, y_mpc, x_mpc] = MPCcontrol(SysD, Np, Nc, Rs, Ru, Nsim, StateEst, Laguerre)
% Calculation of the Augmented State-Space representation:
[SysA] = AugmentedSS(SysD);
% Calculation of the F and Phi matrices, used in the control law:
[PhiPhi, PhiF, PhiR] = MPCmatrices(SysA, Np, Nc);

```

```
Ad = SysD.A; Aa = SysA.A; Bd = SysD.B; Ba = SysA.B;
Cd = SysD.C; Ca = SysA.C; Dd = SysD.D; Da = SysA.D;
n = size(Ad,1); q = size(Aa,1); p = size(Bd,2); m = size(Cd,1);
%Calculations of control input sequence and estimated corresponding output:
if StateEst == 0
    if Laguerre == 0
    % With internal state estimation, incorporated:
    % Initial state and initial augmented state:
    u_mpc   = zeros(1,p); ud = zeros(1,p); % u(k-1) =0
    y_mpc   = zeros(m,1); yd = zeros(m,1);
    x_mpc   = zeros(n,1); xd = zeros(n,1); xa = zeros(q,1);

    for kk=1:Nsim
        DeltaU  = (PhiPhi+Ru)\(PhiR*(Rs(kk,:).')-PhiF*xa);
        ∆u  = DeltaU(1,1); ud = ud+∆u; u_mpc(kk) = ud;
        y_mpc(:, kk) = yd; xd_old = xd; x_mpc(:,kk) = xd;
        xd  = Ad*xd+Bd*ud; yd  = Cd*xd; xa  = [xd-xd_old;yd];
    end
    else
    a     = 0.4; % Pole of DT Laquerre network, scaling factor 0<a<1
    beta  = sqrt(1-a^2); Qc    = diag([0 0 0 0 1 1]); %% C^T*C

    % Create Laguerre Matrix
    [Al,L0T] = lagd(a,Nc); %L(0)^T
    L(:,1)   = L0T; %L(k) = [l1(k), ..., ln(k)]^T and L(k+1) = Al L(k)

    for kk=2:Np; L(:,kk) = Al*L(:,kk-1); end
    psi = 0; omega = 0; % The control law is a function of Psi and Omega

    for i=1:Np
    f   = 0; lag = [];
    for j = 0:i-1; f = f+Aa^(i-1-j)*Ba*L(:,j+1)'; end
    psi = psi+f'*Qc*Aa^i; omega = omega+(f'*Qc*f+Ru);
    end

    % Set the initial values:
    u_mpc = zeros(1,p); u_old = zeros(1,p); % u(k-1) =0
    y_mpc = zeros(m,1); y_current = zeros(m,1);
    x_mpc = zeros(n,1); x_current = zeros(n,1); DeltaX = zeros(n,1);

    for pp=1:Nsim
    DeltaU =  -(L0T.')*(omega\psi) * [DeltaX ; y_current - (Rs(pp,:).')];
    u_current    = u_old + DeltaU; u_mpc(pp) = u_current;
    y_mpc(:, pp) = y_current; x_mpc(:,pp) = x_current;
    x_new   = Ad*x_current + Bd*u_current; y_new  = Cd*x_new;
    DeltaX  = [x_new - x_current];
    u_old   = u_current; y_current = y_new; x_current = x_new;
    end
    end
    else
    if Laguerre == 0
    % State estimation from the Free Energy principle is provided:
    % Initial conditions:
    u_mpc  = zeros(1,p); u0 = zeros(1,p); u=0; % u(k-1) =0
    y_mpc  = zeros(m,1); y0 = zeros(m,1);
    x_mpc  = zeros(n,1); x0 = zeros(n,1); xN0 = zeros(n,1); P0 = 0.1.*eye(n);

    for kk=1:Nsim
        % In case of real system the input will be on the actual set-up,
        % real-time calculations:
        [y1, xN1] = simRealResponse(u0 , xN0);
```

```
        % Calculates the next state estimation, used to caculate the next
        % control input:
        [x1, P1] = MPCFreeEnergyStateEst(SysD, y1, u0, x0, P0);
        Xf = [x1 - x0; y0];
        % Calculates the next control input, PhiPhi can be very large which
        % makes the algorithm large (badly scaled):
        DeltaU = (PhiPhi+Ru)\(PhiR*(Rs(kk,:).')-PhiF*Xf);
        ?u = DeltaU(1,1); u = u+?u; u_mpc(kk) = u;
        % Set the current values equivalent to the past values for the next
        % iteration:
        y0 = y1; xN0 = xN1; x0 = x1; P0 = P1; u0 = u;
        % Calculates the output estimation, with state estimation:
        y_mpc(kk) = Cd*x1;
    end
    else
    a    = 0.4; % Pole of DT Laquerre network, scaling factor 0<a<1
    beta = sqrt(1-a^2); Qc = diag([0 0 0 0 1 1]); %% C^T*C

    % Create Laguerre Matrix
    [Al,L0T] = lagd(a,Nc); %L(0)^T
    L(:,1) = L0T; %L(k) = [l1(k), ..., ln(k)]^T and L(k+1) = Al L(k)

    for kk=2:Np; L(:,kk)=Al*L(:,kk-1); end
    psi = 0; omega = 0; % The control law is a function of Psi and Omega
        for i=1:Np
            f   = 0; lag = [];
            for j=0:i-1; f = f+Aa^(i-1-j)*Ba*L(:,j+1)'; end
            psi = psi+f'*Qc*Aa^i; omega = omega +(f'*Qc*f+Ru);
        end
        % Set the initial values:
        u_mpc = zeros(1,p);  u_old = zeros(1,p); % u(k-1) =0
        y_mpc = zeros(m,1);
        x_mpc = zeros(n,1); xN_old = zeros(n,1); x_old = zeros(n,1);
        P_old = 0.1*eye(n);

        for pp=1:Nsim
            % In case of real system the input will be on the actual
            % set-up, real-time calculations:
            [y_current, xN_current] = simRealResponse(u_old , xN_old);
            % Calculates the next state estimation, used to caculate
            % the next control input:
            [x_current, P_current] = MPCFreeEnergyStateEst(SysD, ...
                                              y_current, u_old, x_old, P_old);
            xa = [x_current - x_old; y_current-(Rs(pp,:).')];
            DeltaU  =  -(L0T.')*(omega\psi) * xa;
            u_current = u_old + DeltaU; u_mpc(pp) = u_current;
            y_mpc(:,pp) = y_current; x_mpc(:,pp) = xN_current;
            xN_old = xN_current; x_old  = x_current;
            P_old  = P_current;u_old  = u_current;
        end
    end
end
end
```

```
function [u_mpc, y_mpc, yN, SysMPC] = MPCcontrolFullLaguerre(Np, Nc, rs, rw, Nsim)
% Set Initial input and output:
u = 0; % u0
% Initial data-store matrices:
y_mpc = zeros(Nsim,1); u_mpc = zeros(Nsim,1);
```

```
% Initial model matrices, ss(A,B,C,D), Discrete-Time:
SysMPC = ss(0.2,0.1,1,0, 0.1);
a = 0.4; % Pole of the DT Laquerre network, scaling factor 0<a<1
beta = sqrt(1-a^2); Qc = 1; %% C^T*C
% Create Laguerre Matrix
[Al,L0T]    = lagd(a,Nc); %L(0)^T
L(:,1) = L0T; %L(k) = [l1(k), l2(k), ..., ln(k)]^T and L(k+1) = Al L(k)

for kk=2:Np; L(:,kk)=Al*L(:,kk-1); end
% State estimation from the Free Energy principle is provided:
% Initial conditions:
y0 = 0; xN0 = 0; P0 = 0; u0 = 0; x0 = 0;
sumx0x0T = 0; sumx0x1T = 0; sumx0u1BT = 0; sumu0u0T = 0;
sumu0x1T = 0; sumu0x1AT = 0;

for kk=1:Nsim
% In case of real system the input will be on the actual set-up,
% real-time calculations:
[y1, xN1] = simRealResponse(u0 , xN0); yN(kk) = y1;
% Calculates the next state estimation, used to caculate the next
% control input:
[x1, P1] = MPCFreeEnergyStateEst(SysMPC, y1, u0, x0, P0);

[muA, muB, sumx0x0T, sumx0x1T, sumx0u1BT, sumu0u0T, sumu0x1T, sumu0x1AT]...
= FreeEnergyModelParametersEstMPC(SysMPC, x0, x1, u0, sumx0x0T, sumx0x1T, ...
                                  sumx0u1BT, sumu0u0T, sumu0x1T, sumu0x1AT);

muA(isnan(muA))=1; muB(isnan(muB))=1;
SysMPC.A = abs(muA); SysMPC.B = abs(muB);
% Calculation of the Augmented State-Space representation:
[SysA] = AugmentedSS(SysMPC); Aa = SysA.A; Ba = SysA.B;
psi = 0; omega = 0; %The control law is a function of Psi and Omega

for i=1:Np
f   = 0; lag = [];
for j=0:i-1; f = f+Aa^(i-1-j)*Ba*L(:,j+1)'; end
psi     = psi+f'*Qc*Aa^i; omega   = omega+(f'*Qc*f+rw);
end
% Calculates the next control input, PhiPhi can be very large which
% makes the algorithm large (badly scaled):
xa = [x1 - x0; y1-rs];
DeltaU  =  -(L0T.')*(omega\psi) * xa; ∆u=DeltaU(1,1); u=u+∆u;

u(u > 2000)     = 2000; u(u < -2000)    = -2000;
u(isnan(u))     = 0; u(¬isfinite(u))= 0;

u_mpc(kk)=u;
% Set the current values equivalent to the past values for next iteration:
y0 = y1; xN0 = xN1; x0 = x1; P0 = P1; u0 = u;
% Calculates the output estimation, with state estimation:
y_mpc(kk) = x1;
end
end
```

```
function [x0, P0] = MPCFreeEnergyStateEst(SysD, y, u, x0, P0)
% Retrieve System model matrices
A = SysD.A; B = SysD.B; C = SysD.C;
n = size(A,1); m = size(C,1);

```

```
% Covariances
sys = load('data.mat');
Q = sys.Q;  R = sys.R;
Q = Q*eye(n); R = R*eye(m);
% Prediction step
x0 = A*x0 + B*u; %Bar notation represents Belief
P0 = A*P0*(A.') + Q;
% Correction step
K = P0*(C.')/(C*P0*(C.')+R);
% Used to correct both the Mean and Variance estimate
x0 = x0 + K*(y-C*x0);
P0 = (eye(n) - K*C)*P0;
end
```

```
function [muA, muB, sumx0x0T, sumx0x1T, sumx0u1BT, sumu0u0T, sumu0x1T, ...
    sumu0x1AT] = FreeEnergyModelParametersEstMPC(sys, mu_x0, mu_x1, u0, ...
                sumx0x0T, sumx0x1T, sumx0u1BT, sumu0u0T, sumu0x1T, sumu0x1AT )
A = sys.A; B = sys.B;
% The estimation of Model matrix A and B
% The Mean:
x0x0T = (mu_x0* (mu_x0.')); sumx0x0T = sumx0x0T + x0x0T;
x0x1T = (mu_x0* (mu_x1.')); sumx0x1T = sumx0x1T + x0x1T;
x0u1BT = (mu_x0*((B*u0).')); sumx0u1BT = sumx0u1BT + x0u1BT;
muA = ((sumx0x1T - sumx0u1BT).')/(sumx0x0T);

u0u0T = (u0*u0); sumu0u0T = sumu0u0T + u0u0T;
u0x1T = (u0* (mu_x1.')); sumu0x1T = sumu0x1T + u0x1T;
u0x1AT = (u0* ((A*mu_x1).')); sumu0x1AT = sumu0x1AT + u0x1AT;
muB = ((sumu0x1T - sumu0x1AT).')/(sumu0u0T);
end
```

BIBLIOGRAPHY

[1] Alexander, S. (2018). God help us, let's try to understand friston on free energy. Slate Star Codex. Retrieved from http://slatestarcodex.com/2018/03/04/god-help-us-lets-try-to-understand-friston-on-free-energy/

[2] Åström, K. J. & Wittenmark, B. (2013). *Computer-controlled systems: Theory and design*. Courier Corporation.

[3] Balaji, B. & Friston, K. (2011). Bayesian state estimation using generalized coordinates. In *Signal processing, sensor fusion, and target recognition xx* (Vol. 8050, 80501Y). International Society for Optics and Photonics.

[4] Beal, M. J. (2003). Variational algorithms for approximate bayesian inference.

[5] Bishop, C. M. (n.d.). *Pattern recognition and machine learning*. Springer-Verlag.

[6] Bohannon, J. (2016). A computer program just ranked the most influential brain scientists of the modern era. ScienceMag. Retrieved from http://www.sciencemag.org/news/2016/11/computer-program-just-ranked-most-influential-brain-scientists-modern-era

[7] Bressler, S. L. (2011). Time series and stochastic processes. Charles E. Schmidt College of Science. Retrieved from http://www.ccs.fau.edu/~bressler/EDU/STSA/Modules/I.pdf

[8] Buckley, C. L., Kim, C. S., McGregor, S., & Seth, A. K. (2017). The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*.

[9] Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

[10] Durieux, B. & Groeneweg, J. (2016). The inverted pendulum; a matter of control.

[11] Friedland, B. (2012). *Control system design: An introduction to state-space methods*. Courier Corporation.

[12] Friston, K. (2008). Hierarchical models in the brain. *PLoS computational biology*, *4*(11), e1000211.

[13] Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, *11*(2), 127.

[14] Friston, K. (2011). What is optimal about motor control? *Neuron*, *72*(3), 488–498.

[15] Friston, K. J., Daunizeau, J., & Kiebel, S. J. (2009). Reinforcement learning or active inference? *PloS one*, *4*(7), e6421.

[16] Friston, K. J., Parr, T., & de Vries, B. (2017). The graphical brain: Belief propagation and active inference. *Network Neuroscience*, *1*(4), 381–414.

[17] Friston, K. J., Trujillo-Barreto, N., & Daunizeau, J. (2008). Dem: A variational treatment of dynamic systems. *Neuroimage*, *41*(3), 849–885.

[18] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., Pezzulo, G. et al. (2016). Active inference and learning. *Neuroscience & Biobehavioral Reviews*, *68*, 862–879.

[19] Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., & Pezzulo, G. (2017). Active inference: A process theory. *Neural computation*, *29*(1), 1–49.

[20] Friston, K., Kilner, J., & Harrison, L. (2006). A free energy principle for the brain. *Journal of Physiology-Paris*, *100*(1-3), 70–87.

[21] Friston, K., Rigoli, F., Ognibene, D., Mathys, C., Fitzgerald, T., & Pezzulo, G. (2015). Active inference and epistemic value. *Cognitive neuroscience*, *6*(4), 187–214.

[22] Friston, K., Samothrakis, S., & Montague, R. (2012). Active inference and agency: Optimal control without cost functions. *Biological cybernetics*, *106*(8-9), 523–541.

[23] Ghahramani, Z. & Hinton, G. E. (1996). *Parameter estimation for linear dynamical systems*. Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science.

[24] Gibson, S. & Ninness, B. (2005). Robust maximum-likelihood estimation of multivariable dynamic systems. *Automatica*, *41*(10), 1667–1682.

[25] Hohwy, J. (2014). Is prediction error minimization all there is to the mind? The brain blogs. Retrieved from http://philosophyofbrains.com/2014/06/22/is-prediction-error-minimization-all-there-is-to-the-mind.aspx

[26] Jing Chen, Y.-F. C., Yulan Huang. (2016). Variational inference i. Cargnegie Mellon University. Retrieved from http://www.cs.cmu.edu/~epxing/Class/10708-16/note/10708_scribe_lecture12.pdf

[27] Karl, F. (2012). A free energy principle for biological systems. *Entropy*, *14*(11), 2100–2121.

[28] Koski, T. (2013). Multivariate gaussian distribution. *Auxiliary Notes for Time Series Analysis SF2943, KTH Royal Institute of Technology, Stockholm, Sweden*.

[29] Likelihood function. (n.d.). https://en.wikipedia.org/wiki/Likelihood_function. Accessed: 2018-11-08.

[30] Ljung, L., Hjalmarsson, H., & Ohlsson, H. (2011). Four encounters with system identification. *European Journal of Control*, *17*(5-6), 449–471.

[31] Luttinen, J. (2013). Fast variational bayesian linear state-space model. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 305–320). Springer.

[32] Maren, A. J. (2017). How to read karl friston (in the original greek). Northwestern University. Retrieved from http://www.aliannajmaren.com/2017/07/27/how-to-read-karl-friston-in-the-original-greek/

[33] Messner, B. & Tilbury, D. (2011). Control tutorials for matlab and simulink: Inverted pendulum. Retrieved November 3, 2018, from http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling

[34] Murphy, K. P. (2013). *Machine learning : A probabilistic perspective*. MIT Press. Retrieved from http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0262018020

[35] Neal, R. M. & Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models* (pp. 355–368). Springer.

[36] Ninness, B. (2009). Some system identification challenges and approaches. *IFAC Proceedings Volumes*, *42*(10), 1–20.

[37] Ojeda, I. B., Tolu, S., Pacheco, M., Christensen, D. J., & Lund, H. H. (2017). A combination of machine learning and cerebellar-like neural networks for the motor control and motor learning of the fable modular robot. *JOURNAL OF ROBOTICS NETWORKING AND ARTIFICIAL LIFE*, *4*(1), 62–66.

[38] Pan, Y., Yan, X., Theodorou, E., & Boots, B. (2016). Adaptive probabilistic trajectory optimization via efficient approximate inference. *arXiv preprint arXiv:1608.06235*.

[39] Recht, B. (2018). The statistical foundations of learning to control. University of Berkeley. Retrieved from https://talks.ox.ac.uk/talks/id/140c6ee7-a00e-4375-9f85-e982d1a073a7/

[40] Rossiter, J. A. (2003). *Model-based predictive control: A practical approach*. CRC press.

[41] Schön, T. (2009). *An explanation of the expectation maximization algorithm*.

[42] Schwarz, W. (2013). The lure of free energy. Retrieved from https://www.umsu.de/wo/2013/600

[43] Sigaud, O. & Peters, J. (2010). From motor learning to interaction learning in robots. In *From motor learning to interaction learning in robots* (pp. 1–12). Springer.

[44] Sjögren, L. (2018). Non-equilibrium stochastic processes in physics, chemistry, and biology. University of Gothenburg. Retrieved from http://physics.gu.se/~frtbm/joomla/index.php?option=com_content&view=article&id=119&Itemid=290

[45] Smith, G. & Robinson, A. (2000). A comparison between the em and subspace identification algorithms for time-invariant linear dynamical systems.
[46] Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, *45*(3), 52–57.
[47] Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT press.
[48] Upton, G. & Cook, I. (2014). *A dictionary of statistics 3e*. Oxford university press.
[49] van den Boom, T. J. & Backx, T. (2010). Model predictive control. *DISC Course, Lecture Notes*, *16*.
[50] Van Overschee, P. & De Moor, B. (1994). N4sid: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, *30*(1), 75–93.
[51] Verhaegen, M. & Verdult, V. (2007). *Filtering and system identification: A least squares approach*. Cambridge university press.
[52] Wang, L. (2009). *Model predictive control system design and implementation using matlab®*. Springer Science & Business Media.
[53] Westwick, D. T. & Kearney, R. E. (2003). *Identification of nonlinear physiological systems*. John Wiley & Sons.
[54] Wolpert, D. M., Ghahramani, Z., & Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, *269*(5232), 1880–1882.
[55] Worden, K. (2016). Is system identification just machine learning? University of Sheffield. Retrieved from http://www.nonlinearbenchmark.org/FILES/SLIDES/keynote_KW.pdf