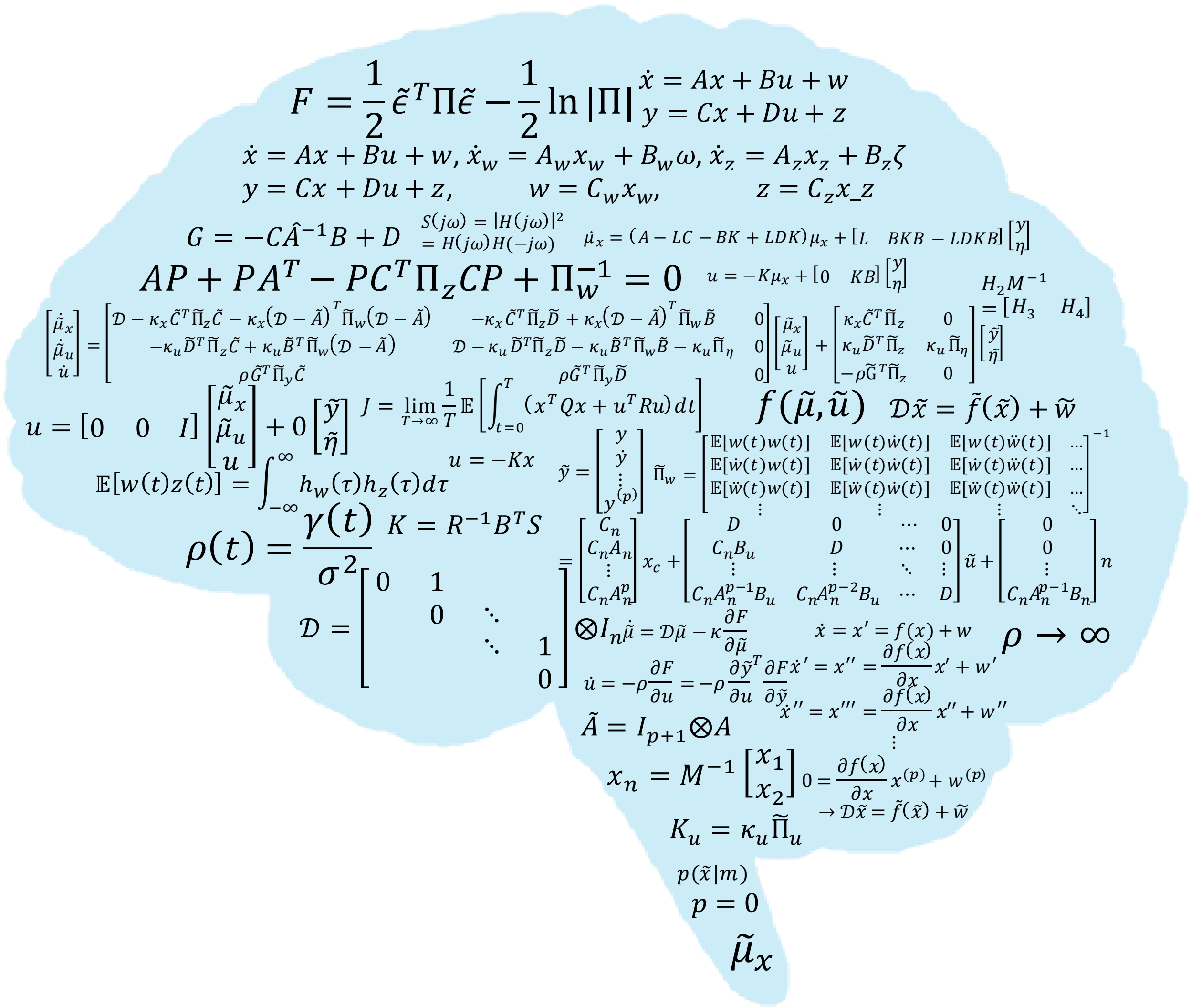
A comparison of Active Inference and Linear-Quadratic Gaussian control

Equivalence and differences for two settings

J.D. Coehoorn

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control and Cognitive Robotics

A comparison of Active Inference and Linear-Quadratic Gaussian control

Equivalence and differences for two settings

Master of Science Thesis

For the double degree of Master of Science in Systems and Control and Mechanical Engineering at Delft University of Technology

J.D. Coehoorn

July 8, 2021

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

Delft University of Technology
Departments of
Delft Center for Systems and Control (dcsc)
and
Cognitive Robotics (CoR)

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

A comparison of Active Inference and Linear-Quadratic Gaussian control

by

Jesse Daniël Coehoorn

in partial fulfillment of the requirements for the degrees of

Master of Science Systems and Control
and
Master of Science Mechanical Engineering

Dated: July 8, 2021

Supervisors: prof.dr.ir. M. Wisse

dr. P. Mohajerin Esfahani

Readers: prof.dr. R. Babuska

A. Anil Meera

Abstract

The Free Energy Principle, which underlies Active Inference (AI), is a way to explain human perception and behaviour. Previous literature has hinted at a relation between AI and Linear-Quadratic Gaussian (LQG) control, the latter being a textbook controller. AI and LQG are, however, defined with different settings in mind: LQG has access to inputs, whereas AI estimates these; LQG is optimal for White Gaussian Noise, whereas noise needs to be coloured for AI, in order to make derivatives. Therefore, a comparison is provided on two bases: the setting of LQG control; and the setting of AI. The optimal LQG controller is obtained for both settings, and AI is applied to both settings as well. When AI is reduced to the setting of LQG, an equivalent expression can be obtained by a proper choice of tuning parameters. This entails choosing a matrix such that the closed-loop is stable, which contrasts LQG control, which is always a stabilizing controller. When LQG is extended closer to the normal AI setting, we find that tuning of AI becomes harder for more complex systems, but that AI is mostly able to show optimal behaviour.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude towards my supervisors Peyman Mohajerin Esfahani and Martijn Wisse for their guidance during the past year. I have learned a lot from both of you. I would like to thank Martijn explicitly for his enthusiastic presentation during the introduction days of the BioMechanical Design track, which got me excited for this topic, and Peyman for his helpful mathematical introductions and discussions. Last, but not least, I would like to thank my wife and my family, who have given me the motivation and encouragement to work on this at home.

Delft, University of Technology
July 8, 2021

J.D. Coehoorn

“There are far, far better things ahead than any we leave behind.”
— *C.S. Lewis*

Chapter 1

Introduction

This first chapter motivates the thesis work. Based on this motivation, a research goal and corresponding research questions are stated. Lastly, the outline of the thesis is provided to accommodate the reader with the structure of the thesis.

1-1 Motivation

For some time now, the Free Energy Principle (FEP), as first introduced by Friston in [12,16], has been around to explain human behaviour and, essentially, life. By way of Dynamic Expectation Maximization (DEM), it provides a mathematical formulation of perceptual inference and learning [17], which can be extended to Active Inference (AI) to also explain action and behaviour [15]. More recently, the FEP and AI have found their way into robotics and control [24–28].

In the paper in which DEM was introduced, Friston et al. claim that DEM and Kalman filtering give the same results in the case that noises are temporally uncorrelated (white), and inputs are not considered [17, p.869]. A similar argument has been given in [2]. This leads to the question whether these results can be extended to the full Kalman filtering setting, i.e. also considering inputs.

Furthermore, in the robotics and control community, the Linear-Quadratic Gaussian (LQG) controller is a textbook controller with a very similar derivation to the Kalman filter. In fact, the Kalman filter is extended with the dual problem - control - and the two resulting parts are put together [1,21,23]. The FEP has a similar intuitive extension from DEM to AI, so it is a very reasonable question to ask whether AI and LQG control are similar or even equivalent, and where similarities or differences come from.

AI and LQG are, however, defined with different settings in mind. As it turns out, when AI is reduced to the same setting that LQG uses, an equivalent expression for the two controllers can be obtained. This is the first time that such a result is shown, and gives an incentive for further study into AI, as LQG is optimal in this setting. When LQG is extended to about the same setting as AI, it is no longer possible to obtain an equivalent representation. We are, however, mostly able to tune AI such that it can show the same, and thus optimal, behaviour as LQG.

1-2 Research goal and questions

Therefore, the research goal is to 'compare AI with LQG control in a theoretical and practical manner'. The theoretical comparison relates to the LQG setting, the practical comparison is done using simulations for the setting that AI uses. To reach this goal, several research questions need to be answered:

1. What are equivalences and differences in the state-space equations of the two controllers?
2. When are the two controllers equivalent?
3. Does the closed-loop of AI on a Linear Time-Invariant (LTI) system depend linearly on the tuning parameters of AI?
4. How does tuned AI compare to LQG control in terms of LQG costs?

The first research question gives us a high-level overview of the differences between the two controllers, which automatically leads to the answer of the second question. This second question is the theoretical part of the comparison. The third question pertains to the tuning of AI: if the closed-loop depends linearly on tuning parameters, there are several tuning procedures readily available, for instance using linear matrix inequalities [5] or the new approach by control as optimization [7, 11]. The last question gives an answer to the practical part of the comparison.

1-3 Outline

First of all, the setting and common concepts are introduced in chapter 2. When introducing the setting, it is made clear that two comparisons are needed. LQG control is introduced for both settings of the two comparisons, after which the basics of AI are presented. Starting from chapter 3, we obtain new results. In this chapter, the equations and state-space representations for AI on a LTI system for different formulations of the Free Energy (FE) (needed for the two comparisons) are provided. In chapter 4, the theoretical and practical comparison is done, using simulations in MATLAB for the practical part. Lastly, in chapter 5, the conclusions are summarized and recommendations for further research are given.

At the end of the document, the appendices together with a list of acronyms and list of symbols can be found.

Chapter 2

Preliminaries

First, concepts common to both Active Inference (AI) and the Linear-Quadratic Gaussian (LQG) problem are introduced. In order to facilitate a comparison, we make sure that we use the same symbols for the same concepts. We also highlight the different settings between the two controllers and show how this will affect the comparison. After that, the standard infinite-horizon LQG controller is shortly introduced, consisting of the Kalman-Bucy filter and the Linear-Quadratic Regulator (LQR). Additionally, a situation is shown where the Kalman filter problem is singular, and the optimal solution to this new problem is given. Lastly, AI is introduced for general systems; in the next chapter this will be applied to Linear Time-Invariant (LTI) systems.

2-1 Setting

The setting that will be considered in this thesis work is a controllable and observable LTI dynamical system that will be controlled using either LQG or AI. LQG is defined on linear systems [1], in contrast to AI, which can be used on arbitrary dynamical systems [15]. The LTI system is defined as follows:

$$\begin{aligned} \dot{x} &= Ax + Bu + w, \\ y &= Cx + Du + z, \end{aligned} \tag{2-1}$$

where $x \in \mathbb{R}^n$ are the states of the system, $u \in \mathbb{R}^m$ are the inputs to the system and $y \in \mathbb{R}^q$ are the outputs, observations or measurements available to the controller. $w \in \mathbb{R}^n$ is the process noise and $z \in \mathbb{R}^q$ is the measurement noise, which are both additive and zero-mean Gaussian distributed. Additionally, we have the state matrix A, input matrix B, output matrix C and D, the feedthrough matrix.

To make a comparison possible, we thus have to derive AI on linear systems, which will be done in chapter 3. Furthermore, the inputs are known to the standard LQG controller, whereas AI estimates the inputs, eliminating the need to feed inputs back to the controller.

So, in order to do a comparison on the level of LQG, AI should be formulated to include knowledge of inputs, which is done in subsection 3-1-3. In subsection 3-1-2, a connection is made to some existing literature, which might be seen as an intermediate step between no knowledge and full knowledge of inputs.

Lastly, LQG control is optimal when the noises w and z are white, leading to White Gaussian Noise (WGN), with covariance matrices $\mathbb{E}[w(t)w(t+\tau)] = \Pi_w^{-1}\delta(\tau)$ and $\mathbb{E}[z(t)z(t+\tau)] = \Pi_z^{-1}\delta(\tau)$, respectively, where $\delta(t)$ denotes the Dirac delta function [1, p.180]. AI, on the other hand, needs the noises to be dependent in time, such that derivatives of the outputs can be taken. In Table 2-1, an overview of the settings is provided.

Table 2-1: Overview of the settings for AI and LQG.

	AI	LQG
Controlled system	Arbitrary	Linear
Knowledge of inputs	Unknown (estimates)	Known
Noise distribution	Gaussian	Gaussian
Noise time-dependency	Dependent	Independent (white)

There are now two main ways to deal with the various differences and do the comparison. Firstly, we can reduce AI to the LQG setting, where we include knowledge of inputs and say that the noises are WGN. Secondly, we can extend the LQG problem by assuming that the noises w and z are given by filtering WGN with strictly proper LTI systems. This leads to the description

$$\begin{aligned} \dot{x} &= Ax + Bu + w, & \dot{x}_w &= A_w x_w + B_w \omega, & \dot{x}_z &= A_z x_z + B_z \zeta, \\ y &= Cx + Du + z, & w &= C_w x_w, & z &= C_z x_z, \end{aligned} \tag{2-2}$$

where ω and ζ are normalized WGN, which means that $\Pi_\omega = \Pi_\zeta = I$. Both controllers need to be specified for both comparisons, so we give an overview of the two comparisons and their relevant chapters:

- Comparison I: Normal LQG and reduced AI
  WGN setting with knowledge of inputs, performed in section 4-2. Relevant sections are subsection 2-2-1, section 2-3 and subsection 3-1-3 and are marked with I. This leads to an equivalent expression for the two controllers.

- Comparison II: Extended LQG and normal AI
  Linearly filtered WGN, without knowledge of inputs for AI, performed in section 4-3. Relevant sections are section 2-2, section 2-3 and subsection 3-1-1 and are marked with II. An equivalent expression is no longer possible, but AI can mostly be tuned to obtain the same (optimal) LQG costs. This comparison is performed numerically, using MATLAB.

2-2 Introduction to LQG control

First, we show the standard infinite-horizon LQG problem for LTI systems, resulting in the main controller equation to be used for the first comparison. After that, a solution is given to obtain a LQG controller in the case of a singular covariance for the measurement noise, resulting in MATLAB code to be used in the second comparison.

2-2-1 I: The standard infinite-horizon LQG problem

The book by Anderson and Moore [1] is used as main reference for this section. In the standard infinite-horizon LQG problem (for LTI systems), we want to control the system as in Equation 2-1, with the noises w and z being WGN. For simplicity, we assume that the cross-covariance is 0. The LQG problem is now to control this system, such that the cost

$$J = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\int_{t=0}^{T} \left(x^\top Qx + u^\top Ru\right) dt\right] \tag{2-3}$$

is minimized. The matrix $Q \geq 0$ gives a weighing on the states x, the matrix $R > 0$ a weighing on the inputs u. Sometimes, a cross-term $2x^\top Nu$ is added, but is often omitted; it can be made zero by doing an appropriate coordinate change.

The optimal controller consists of two independently designed parts, thanks to the separation principle [1, p.218]. The first part is a filter, which gives optimal (unbiased, minimal variance) estimates μ_x of the states x [23]. This filter is given by the equation

$$\dot{\mu}_x = A\mu_x + Bu + L(y - C\mu_x - Du), \tag{2-4}$$

where L is the Kalman gain, calculated with $L = PC^\top\Sigma_y^{-1}$. Here we have exchanged the normally used $\hat{x}$ with μ_x, to make the comparison with AI more clear. The matrix P in this equation is the error covariance matrix, which is the solution to the Algebraic Riccati Equation (ARE)

$$AP + PA^\top - PC^\top\Pi_z CP + \Pi_w^{-1} = 0.$$

The second part is given by the feedback law $u = -Kx$, where $K = R^{-1}B^\top S$ is the optimal control feedback gain from the LQR problem, which is the dual problem to the filter problem [21, 22]. The matrix S is again the solution to an ARE, given by

$$A^\top S + SA - SBR^{-1}B^\top S + Q = 0.$$

The optimal controller is now simply given by using the feedback law on the state estimates, resulting in

$$\begin{aligned}\dot{\mu}_x &= (A - LC - BK + LDK)\mu_x + Ly,\\ u &= -K\mu_x.\end{aligned}$$

A common practice is to add a feed-forward term if the states need to be steered to some non-zero reference point $B\eta$, where η is the equilibrium input for the reference point. This feed-forward term is given by $KB\eta$, resulting in the controller

$$\begin{aligned}\dot{\mu}_x &= (A - LC - BK + LDK)\mu_x + \begin{bmatrix} L & BKB - LDKB \end{bmatrix}\begin{bmatrix} y \\ \eta \end{bmatrix},\\ u &= -K\mu_x + \begin{bmatrix} 0 & KB \end{bmatrix}\begin{bmatrix} y \\ \eta \end{bmatrix}.\end{aligned} \tag{2-5}$$

This controller equation is the main equation to be used in the first comparison and now minimizes the cost function

$$J = \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\int_0^T \Big((x - B\eta)^\top Q(x - B\eta) + u^\top Ru\Big)\right].$$

2-2-2 II: LQG control for a singular Kalman filter

In order to determine the LQG controller for the systems in Equation 2-2 so that we can do the second comparison, we write them as one extended system:

$$\begin{aligned}\begin{bmatrix}\dot{x}\\ \dot{x}_w\\ \dot{x}_z\end{bmatrix} &= \underbrace{\begin{bmatrix}A & C_w & 0\\ 0 & A_w & 0\\ 0 & 0 & A_z\end{bmatrix}}_{A_n}\underbrace{\begin{bmatrix}x\\ x_w\\ x_z\end{bmatrix}}_{x_n} + \underbrace{\begin{bmatrix}B\\ 0\\ 0\end{bmatrix}}_{B_u} u + \underbrace{\begin{bmatrix}0 & 0\\ B_w & 0\\ 0 & B_z\end{bmatrix}}_{B_n}\underbrace{\begin{bmatrix}\omega\\ \zeta\end{bmatrix}}_{n},\\ y &= \underbrace{\begin{bmatrix}C & 0 & C_z\end{bmatrix}}_{C_n}\begin{bmatrix}x\\ x_w\\ x_z\end{bmatrix} + Du.\end{aligned} \tag{2-6}$$

Because y now no longer directly depends on WGN, because the WGN first passes through a filter, the matrix Π_z^{-1} is singular. This also means that the Kalman problem is singular. In order to solve this issue, we use the technique from [6] and extend it to also include inputs. The main part of the technique is to take derivatives of the out- and inputs, which is similar to AI, as will become clear later when AI is introduced.

Derivatives are taken until the WGN appears, and can be computed as follows:

$$\tilde{y} = \begin{bmatrix}y\\ \dot{y}\\ \vdots\\ y^{(p)}\end{bmatrix} = \begin{bmatrix}C_n\\ C_nA_n\\ \vdots\\ C_nA_n^p\end{bmatrix} x_n + \begin{bmatrix}D & 0 & \cdots & 0\\ C_nB_u & D & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ C_nA_n^{p-1}B_u & C_nA_n^{p-2}B_u & \cdots & D\end{bmatrix}\tilde{u} + \begin{bmatrix}0\\ C_nB_n\\ \vdots\\ C_nA_n^{p-1}B_n\end{bmatrix} n.$$

Of course, if the system has multiple outputs ($q > 1$), it could be that some outputs can be differentiated more times than other outputs. To overcome this issue, one can simply calculate the derivatives for each row of C_n separately. In this work, we assume that all the outputs can be differentiated the same amount of times. After differentiating, the outputs and its derivatives are decomposed into two vectors: x_2, which is noise-free, and z, which includes a white noise term:

$$\begin{bmatrix}x_2\\ z\end{bmatrix} = \tilde{y} - G\tilde{u} = \begin{bmatrix}M_2\\ H_2\end{bmatrix} x_n + \begin{bmatrix}0\\ D_2\end{bmatrix} n.$$

We now actually have that x_2 are 'perfect' measurements of linear combinations of x_n, so it is only necessary to estimate x_1, which is linearly independent of x_2:

$$\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}M_1\\ M_2\end{bmatrix} x_n = Mx_n,$$

which means that $x_n = M^{-1}\begin{bmatrix}x_1\\x_2\end{bmatrix}$.

This is inserted into the original equation $\dot{x}_n = A_n x_n + B_u u + B_n n$ to obtain

$$\begin{aligned}\begin{bmatrix}\dot{x}_1\\\dot{x}_2\end{bmatrix} &= M\dot{x}_n\\ &= MA_nM^{-1}\begin{bmatrix}x_1\\x_2\end{bmatrix} + MB_u u + MB_n n\\ &= \begin{bmatrix}F_{11} & F_{12}\\F_{21} & F_{22}\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}B_{u,1}\\B_{u,2}\end{bmatrix}u + \begin{bmatrix}B_{n,1}\\B_{n,2}\end{bmatrix}n.\end{aligned}$$

Remember that for the second row of equations we have perfect measurements, so only the first row needs to be estimated with a Kalman filter. To do this, we can make use of the 'measurements' z, but this needs to be transformed such that x_2 is no longer part of the measurements, by setting $H_2M^{-1} = \begin{bmatrix}H_3 & H_4\end{bmatrix}$, resulting in:

$$\begin{aligned}z &= H_2x_n + D_2n\\ &= H_2M^{-1}\begin{bmatrix}x_1\\x_2\end{bmatrix} + D_2n\\ &= H_3x_1 + H_4x_2 + D_2n.\end{aligned}$$

The outputs which can be used for the Kalman filter on x_1 can now be defined as:

$$\begin{aligned}y' &= z - H_4x_2\\ &= H_3x_1 + D_2n.\end{aligned}$$

Now, the normal Kalman filter

$$\begin{aligned}\dot{\mu}_{x_1} &= F_{11}\mu_{x_1} + \begin{bmatrix}B_{u,1} & F_{12}\end{bmatrix}\begin{bmatrix}u\\x_2\end{bmatrix} + K(y' - H_3\mu_{x_1}),\\ \mu_{x_n} &= M^{-1}\begin{bmatrix}\mu_{x_1}\\x_2\end{bmatrix},\end{aligned}$$

can simply be used in combination with the LQR controller, to obtain the optimal LQG controller for the problem as posed in the beginning. In section A-1, a MATLAB implementation for this can be found, which will be used for the second comparison.

2-3 I&II: Introduction to Active Inference

We will now shortly introduce AI on general systems. In chapter 3, this will be elaborated for LTI systems for both settings that will be used in the comparison, but the general machinery is introduced here. Only concepts that are relevant for this thesis are introduced, referring the reader to [8, 15] for further reference. We will start by explaining generalised coordinates and generalised processes, after which AI itself will be described.

A generalised coordinate consists of the coordinate itself and its time derivatives. We denote any generalisation with a tilde, e.g. for the states x of a system, this becomes $\tilde{x} = [x, x', x'', \dots]^\top$ [13]. Generally, we only use the first p derivatives of such a coordinate. This concept can also be applied to processes or systems. We consider a system m with the state evolution equation $\dot{x} = f(x) + w$ and measurement equation $y = g(x) + z$ and only consider the first p derivatives of this system. If we assume local linearity, we can write the generalised process as [13]:

$$\begin{aligned}
\dot{x} &= x' = f(x) + w & y &= g(x) + z\\
\dot{x}' &= x'' = \frac{\partial f(x)}{\partial x}x' + w' & y' &= \frac{\partial g(x)}{\partial x}x' + z'\\
\dot{x}'' &= x''' = \frac{\partial f(x)}{\partial x}x'' + w'' & y'' &= \frac{\partial g(x)}{\partial x}x'' + z''\\
&\vdots & &\vdots\\
0 &= \frac{\partial f(x)}{\partial x}x^{(p)} + w^{(p)} & y^{(p)} &= \frac{\partial g(x)}{\partial x}x^{(p)} + z^{(p)}
\end{aligned}$$

As can be seen, a generalised process consists of the actual process and its linearised time-derivatives, and we can write that $\mathcal{D}\tilde{x} = \tilde{f}(\tilde{x}) + \tilde{w}$ and $\tilde{y} = \tilde{g}(\tilde{x}) + \tilde{z}$, with $\mathcal{D}$ a derivative operator for the generalized coordinates, such that $\tilde{x}' = \mathcal{D}\tilde{x}$. $\mathcal{D}$ is thus a block matrix with identity blocks on the super-diagonal and zeros otherwise:

$$\mathcal{D} = \begin{bmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix} \otimes I_n.$$

$\otimes$ is the Kronecker product and I_n is an identity matrix of size n. This generalised process can be extended to include inputs in the same manner, i.e. we obtain $\mathcal{D}\tilde{x} = \tilde{f}(\tilde{x}, \tilde{u}) + \tilde{w}$ and $\tilde{y} = \tilde{g}(\tilde{x}, \tilde{u}) + \tilde{z}$. These equations are called the generative process.

Now, let us introduce AI. AI is a consequence of the Free Energy Principle (FEP), which starts with the premise that a biological agent, or system m, needs to be within some set of states to exist or survive. Bacteria, for example, have to be in a chemical mixture with a certain concentration and within some temperature range. Using an *ensemble density* on generalised states $p(\tilde{x}|m)$, we can denote the probability distribution of the generalized states (the trajectory) of the system. These states evolve according to the generative process. Now, to adhere to the FEP, this ensemble density should have a narrow distribution or concentrated mass, which corresponds to having low entropy. Using information theoretics and under some assumptions, we can provide an upper bound on the entropy, which is given by the Free Energy (FE) [8,15].

The expression for the FE F is

$$\begin{aligned}
F &= \frac{1}{2}\tilde{\epsilon}^\top \Pi \tilde{\epsilon} - \frac{1}{2}\ln|\Pi|,\\
\tilde{\epsilon} &= \begin{bmatrix} \tilde{\epsilon}_y = \tilde{y} - g(\tilde{\mu}) \\ \tilde{\epsilon}_x = \mathcal{D}\tilde{\mu}_x - f(\tilde{\mu}) \\ \tilde{\epsilon}_u = \tilde{\mu}_u - \tilde{\eta} \end{bmatrix},
\end{aligned} \tag{2-7}$$

where $f(\tilde{\mu})$ and $g(\tilde{\mu})$ are generative *models* for the states and outputs, respectively. These denote our belief or desired belief of the generative process as specified by $\tilde{f}(\tilde{x}, \tilde{u})$ and $\tilde{g}(\tilde{x}, \tilde{u})$. $\tilde{\mu} = \{\tilde{\mu}_x, \tilde{\mu}_u\}$ are the generalized estimates of states and inputs, respectively, and $\tilde{\eta}$ the generalized prior belief on the inputs. This can also be thought of as a generalised reference signal in the same way as in Equation 2-5. Π is a block diagonal precision matrix with output noise, system noise and input prior precisions, $\tilde{\Pi}_z$, $\tilde{\Pi}_w$ and $\tilde{\Pi}_u$, respectively.

So, if a system is to adhere to the FEP, it should minimize its entropy. This can be achieved by minimizing the FE. This is usually done employing a generalised gradient descent on the estimates $\tilde{\mu}$, which constitutes Dynamic Expectation Maximization (DEM) [17]. To obtain AI, we also do a regular gradient descent on the inputs, by noting that $\tilde{y}$ is influenced by u [15]:

$$\begin{aligned} \dot{\tilde{\mu}} &= \mathcal{D}\tilde{\mu} - \kappa \frac{\partial F}{\partial \tilde{\mu}}, \\ \dot{u} &= -\rho \frac{\partial F}{\partial u} = -\rho \frac{\partial \tilde{y}}{\partial u}^{\top} \frac{\partial F}{\partial \tilde{y}}, \end{aligned} \tag{2-8}$$

with κ and ρ the learning rates. These gradient descent equations, together with Equation 2-7, form the main equations to be used in the next chapter.

Chapter 3

Active Inference on LTI systems

In this chapter, the Active Inference (AI) controller on a Linear Time-Invariant (LTI) system is derived. This is done for three different formulations of the Free Energy (FE), each resulting in a different controller, the first and the last of which are necessary for the two comparisons. After that, the closed-loop equations for the first formulation are given, to obtain an answer to the third research question. Next, two ways to obtain the precision matrices for linearly filtered White Gaussian Noise (WGN) are showed, together with some examples. Lastly, a discussion on the precision matrix for plain WGN is provided.

3-1 AI controller for different formulations of the FE

In this section, we will specify the AI controller on LTI systems. For both of the settings on which the two comparisons will be performed, we start with the FE Equation 2-7, and obtain the controller using the gradient descents in Equation 2-8. The generative model that we use, mirrors the generative process (Equation 2-1) and is given by:

$$\begin{aligned} f(\tilde{\mu}) &= \tilde{A}\tilde{\mu}_x + \tilde{B}\tilde{\mu}_u, \\ g(\tilde{\mu}) &= \tilde{C}\tilde{\mu}_x + \tilde{D}\tilde{\mu}_u, \end{aligned}$$

with $\tilde{A} = I_{p+1} \otimes A$, $\tilde{B} = I_{p+1} \otimes B$, $\tilde{C} = I_{p+1} \otimes C$ and $\tilde{D} = I_{p+1} \otimes D$.

In the original literature on AI, the FE expression as given by Equation 2-7 is used. This expression will thus be used in the second comparison. There are, however, some alterations possible. In [18] and [4], the implicit assumption is made that the estimates of the inputs $\tilde{\mu}_u$ are equal to the priors on the inputs $\tilde{\eta}$. This can be represented by setting $\tilde{\Pi}_u = \lambda I$, and letting $\lambda \to \infty$. This case is quickly presented, but not further investigated.

One can also assume that the inputs are directly known, eliminating the need for estimating the inputs. In most control problems, this is actually the case. This is also more similar to the Linear-Quadratic Gaussian (LQG) control problem, where this assumption is also made. This formulation will thus be used in the first comparison, the one on the grounds of LQG. This is a new approach within AI, which has not been investigated before.

3-1-1 II: Original formulation

Writing out the expressions from Equation 2-7, we get the following FE equation:

$$F(\tilde{\mu},\tilde{y},\tilde{\eta}) = \frac{1}{2}(\tilde{y}-g(\tilde{\mu}))^\top \tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu}))+\frac{1}{2}(\mathcal{D}\tilde{\mu}_x-f(\tilde{\mu}))^\top \tilde{\Pi}_w(\mathcal{D}\tilde{\mu}_x-f(\tilde{\mu}))+\frac{1}{2}(\tilde{\mu}_u-\tilde{\eta})^\top \tilde{\Pi}_u(\tilde{\mu}_u-\tilde{\eta}).$$

Note that this FE is convex[1] for linear system and that we omitted the constant $\ln|\Pi|$ term, as it is not relevant for the subsequent derivations. This constant term will also be left out in the rest of the thesis work.

The partial derivatives of the FE are then simply:

$$\begin{aligned}
\frac{\partial F}{\partial \tilde{\mu}_x} &= -\tilde{C}^\top \tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu})) + (\mathcal{D}-\tilde{A})^\top \tilde{\Pi}_w(\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu})),\\
\frac{\partial F}{\partial \tilde{\mu}_u} &= -\tilde{D}^\top \tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu})) - \tilde{B}^\top \tilde{\Pi}_w(\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu})) + \tilde{\Pi}_u(\tilde{\mu}_u-\tilde{\eta}),\\
\frac{\partial F}{\partial u} &= \left(\frac{\partial \tilde{y}}{\partial u}\right)^\top \tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu})) = \tilde{G}^\top \tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu})).
\end{aligned}$$

The forward model $\frac{\partial \tilde{y}}{\partial u} = \tilde{G}$, which denotes how the observations depend on the inputs, is given by $\tilde{G}^\top = \begin{bmatrix} G^\top & 0 & \cdots & 0\end{bmatrix} = \begin{bmatrix} (-C\hat{A}^{-1}B + D)^\top & 0 & \cdots & 0\end{bmatrix}$, following the derivations in [18, 20]. This is the steady-state gain matrix for a system with matrices $\hat{A}$, B, C and D, where $\hat{A}$ is Hurwitz. If one wants to best model the system, $\hat{A}$ should be equal to A. In some applications in robotics, just the identity matrix is used, letting the difference be resolved by proper tuning of the control learning rate; see for example [3, 26].

Finally, the controller is obtained by employing the generalised gradient descent for the estimates $\tilde{\mu}$ and a normal gradient descent on the inputs u, as in Equation 2-8, after which the output of the controller is the input state:

$$\begin{aligned}
&\begin{bmatrix}\dot{\tilde{\mu}}_x\\ \dot{\tilde{\mu}}_u\\ \dot{u}\end{bmatrix} = \\
&\begin{bmatrix}
\mathcal{D} - \kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{C} - \kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & -\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{D} + \kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w\tilde{B} & 0\\
-\kappa_u\tilde{D}^\top\tilde{\Pi}_z\tilde{C} + \kappa_u\tilde{B}^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & \mathcal{D} - \kappa_u\tilde{D}^\top\tilde{\Pi}_z\tilde{D} - \kappa_u\tilde{B}^\top\tilde{\Pi}_w\tilde{B} - \kappa_u\tilde{\Pi}_u & 0\\
\rho\tilde{G}^\top\tilde{\Pi}_z\tilde{C} & \rho\tilde{G}^\top\tilde{\Pi}_z\tilde{D} & 0
\end{bmatrix}
\begin{bmatrix}\tilde{\mu}_x\\ \tilde{\mu}_u\\ u\end{bmatrix}\\
&\quad + \begin{bmatrix}\kappa_x\tilde{C}^\top\tilde{\Pi}_z & 0\\ \kappa_u\tilde{D}^\top\tilde{\Pi}_z & \kappa_u\tilde{\Pi}_u\\ -\rho\tilde{G}^\top\tilde{\Pi}_z & 0\end{bmatrix}\begin{bmatrix}\tilde{y}\\ \tilde{\eta}\end{bmatrix},\\
&\quad u = \begin{bmatrix}0 & 0 & I\end{bmatrix}\begin{bmatrix}\tilde{\mu}_x\\ \tilde{\mu}_u\\ u\end{bmatrix},
\end{aligned} \tag{3-1}$$

with κ_x the state estimation learning rate, κ_u the input estimation learning rate and ρ the control learning rate. These matrix learning rates are the tuning parameters of the AI controller. The precision matrix of the input prior $\tilde{\Pi}_u$ can also be considered to be a tuning

[1]In outputs $\tilde{y}$, estimates $\tilde{\mu}$ and priors $\tilde{\eta}$.

parameter, as it can be freely chosen. In section A-2, a MATLAB implementation for the closed-loop of AI on a LTI system can be found, where use is made of MATLAB's Control System Tuner toolbox. This code is used in the second comparison.

3-1-2 Inputs are equal to the priors

When we set $\tilde{\mu}_u = \tilde{\eta}$, which is equivalent to the limiting case of $\tilde{\Pi}_u = \lambda I$, with $\lambda \to \infty$, the controller obtained from the original formulation reduces to:

$$\begin{aligned}
\begin{bmatrix} \dot{\tilde{\mu}}_x \\ \dot{u} \end{bmatrix} &= \begin{bmatrix} \mathcal{D} - \kappa_x \tilde{C}^\top \tilde{\Pi}_z \tilde{C} - \kappa_x (\mathcal{D} - \tilde{A})^\top \tilde{\Pi}_w (\mathcal{D} - \tilde{A}) & 0 \\ \rho \tilde{G}^\top \tilde{\Pi}_z \tilde{C} & 0 \end{bmatrix} \begin{bmatrix} \tilde{\mu}_x \\ u \end{bmatrix} \\
&+ \begin{bmatrix} \kappa_x \tilde{C}^\top \tilde{\Pi}_z & -\kappa_x \tilde{C}^\top \tilde{\Pi}_z \tilde{D} + \kappa_x (\mathcal{D} - \tilde{A})^\top \tilde{\Pi}_w \tilde{B} \\ -\rho \tilde{G}^\top \tilde{\Pi}_z & \rho \tilde{G}^\top \tilde{\Pi}_z \tilde{D} \end{bmatrix} \begin{bmatrix} \tilde{y} \\ \tilde{\eta} \end{bmatrix}, \\
u &= \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} \tilde{\mu}_x \\ u \end{bmatrix},
\end{aligned}$$

which is a more complete version of the controller found in [18]. A MATLAB implementation for this can also be found in section A-2.

3-1-3 I: Inputs are known

Instead of estimating the inputs, when we know the inputs exactly, like in most control systems, we can replace the FE with

$$\begin{aligned}
F(\tilde{\mu}_x, \tilde{y}, \tilde{\eta}, \tilde{u}) = &\frac{1}{2}(\tilde{y} - g(\tilde{\mu}_x, \tilde{u}))^\top \tilde{\Pi}_z (\tilde{y} - g(\tilde{\mu}_x, \tilde{u})) + \frac{1}{2}(\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu}_x, \tilde{u}))^\top \tilde{\Pi}_w (\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu}_x, \tilde{u})) \\
&+ \frac{1}{2}(\tilde{u} - \tilde{\eta})^\top \tilde{\Pi}_u (\tilde{u} - \tilde{\eta}).
\end{aligned}$$

This is closer to the setting on which LQG is defined, and the results from this section will be used in the first comparison.

This FE leads to the partial derivatives:

$$\begin{aligned}
\frac{\partial F}{\partial \tilde{\mu}_x} &= -\tilde{C}^\top \tilde{\Pi}_z (\tilde{y} - g(\tilde{\mu}_x, \tilde{u})) + (\mathcal{D} - \tilde{A})^\top \tilde{\Pi}_w (\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu}_x, \tilde{u})), \\
\frac{\partial F}{\partial u} &= \left(\tilde{G}^\top - \begin{bmatrix} D^\top & 0 & \cdots & 0 \end{bmatrix}\right) \tilde{\Pi}_z (\tilde{y} - g(\tilde{\mu}_x, \tilde{u})) \\
&- \begin{bmatrix} B^\top & 0 & \cdots & 0 \end{bmatrix} \tilde{\Pi}_w (\mathcal{D}\tilde{\mu}_x - f(\tilde{\mu}_x, \tilde{u})) \\
&+ I_{m \times m(p+1)} \tilde{\Pi}_u (\tilde{u} - \tilde{\eta}),
\end{aligned}$$

with $I_{m \times m(p+1)}$ an identity matrix concatenated with zeros in the larger direction.

The controller then becomes

$$\begin{aligned}\begin{bmatrix}\dot{\tilde{\mu}}_x\\ \dot{u}\end{bmatrix} &= \begin{bmatrix}\mathcal{D}-\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{C}-\kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & -\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{D}_1+\kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w\tilde{B}_1\\ -\rho\begin{bmatrix}(C\hat{A}^{-1}B)^\top & 0 & \cdots & 0\end{bmatrix}\tilde{\Pi}_z\tilde{C}+\rho B_1^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & -\rho\begin{bmatrix}(C\hat{A}^{-1}B)^\top & 0 & \cdots & 0\end{bmatrix}\tilde{\Pi}_z\tilde{D}_1-\rho B_1^\top\tilde{\Pi}_w\tilde{B}_1-\rho I_{m\times m(p+1)}\tilde{\Pi}_u I_{m(p+1)\times m}\end{bmatrix}\begin{bmatrix}\tilde{\mu}_x\\ u\end{bmatrix}\\ &+\begin{bmatrix}\kappa_x\tilde{C}^\top\tilde{\Pi}_z & 0 & -\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{D}_2+\kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w\tilde{B}_2\\ \rho\begin{bmatrix}(C\hat{A}^{-1}B)^\top & 0 & \cdots & 0\end{bmatrix}\tilde{\Pi}_z & \rho I_{m\times m(p+1)}\tilde{\Pi}_u & -\rho\begin{bmatrix}(C\hat{A}^{-1}B)^\top & 0 & \cdots & 0\end{bmatrix}\tilde{\Pi}_z\tilde{D}_2-\rho B_1^\top\tilde{\Pi}_w\tilde{B}_2-\rho I_{m\times m(p+1)}\tilde{\Pi}_u\begin{bmatrix}0\\ I_{mp}\end{bmatrix}\end{bmatrix}\begin{bmatrix}\tilde{y}\\ \tilde{\eta}\\ \tilde{u}_2\end{bmatrix},\\ u &= \begin{bmatrix}0 & I\end{bmatrix}\begin{bmatrix}\tilde{\mu}_x\\ u\end{bmatrix},\end{aligned}$$

where B_1 is the first block column of $\tilde{B}$ and B_2 are the remaining columns; similarly for D_1 and D_2. $\tilde{u}_2$ is given by $\tilde{u}$, excluding u.

Alternatively, one can take the partial derivative with respect to $\tilde{u}$ to obtain

$$\frac{\partial F}{\partial \tilde{u}} = (\tilde{H}-\tilde{D})^\top\tilde{\Pi}_z(\tilde{y}-g(\tilde{\mu}_x,\tilde{u}))-\tilde{B}^\top\tilde{\Pi}_w(\mathcal{D}\tilde{\mu}_x-f(\tilde{\mu}_x,\tilde{u}))+\tilde{\Pi}_u(\tilde{u}-\tilde{\eta}),$$

with

$$\tilde{H}=\begin{bmatrix}-C\hat{A}^{-1}B+D & -C\hat{A}^{-2}B & \cdots & -C\hat{A}^{-p}B & -C\hat{A}^{-p-1}B\\ 0 & -C\hat{A}^{-1}B+D & \cdots & -C\hat{A}^{-p+1}B & -C\hat{A}^{-p}B\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & \cdots & -C\hat{A}^{-1}B+D & -C\hat{A}^{-2}B\\ 0 & 0 & \cdots & 0 & -C\hat{A}^{-1}B+D\end{bmatrix},$$

which is derived in [20], which results in the controller

$$\begin{aligned}\begin{bmatrix}\dot{\tilde{\mu}}_x\\ \dot{\tilde{u}}\end{bmatrix} &= \begin{bmatrix}\mathcal{D}-\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{C}-\kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & -\kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{D}+\kappa_x(\mathcal{D}-\tilde{A})^\top\tilde{\Pi}_w\tilde{B}\\ \rho(\tilde{H}-\tilde{D})^\top\tilde{\Pi}_z\tilde{C}+\rho\tilde{B}^\top\tilde{\Pi}_w(\mathcal{D}-\tilde{A}) & \rho(\tilde{H}-\tilde{D})^\top\tilde{\Pi}_z\tilde{D}-\rho\tilde{B}^\top\tilde{\Pi}_w\tilde{B}-\rho\tilde{\Pi}_u\end{bmatrix}\begin{bmatrix}\tilde{\mu}_x\\ \tilde{u}\end{bmatrix}\\ &+\begin{bmatrix}\kappa_x\tilde{C}^\top\tilde{\Pi}_z & 0\\ -\rho(\tilde{H}-\tilde{D})^\top\tilde{\Pi}_z & \rho\tilde{\Pi}_u\end{bmatrix}\begin{bmatrix}\tilde{y}\\ \tilde{\eta}\end{bmatrix},\\ u &= \begin{bmatrix}0 & I_{m\times m(p+1)}\end{bmatrix}\begin{bmatrix}\tilde{\mu}_x\\ \tilde{u}\end{bmatrix}.\end{aligned}$$

As it turns out, in the first comparison, both of these controllers reduce to the same expression for $p=0$ (the WGN case).

3-2 Closed-loop

In this section, we will answer the research question:

Does the closed-loop of AI on a LTI system depend linearly on the tuning parameters of AI?

To be able to determine this answer, we obtain the closed-loop system description for the original implementation of AI on the systems in Equation 2-2. If the answer is positive, we can make use of several tuning methods available to optimally tune the AI controller for the second part of the comparison.

Because of the derivatives of y, it is not easy to directly calculate the state-space representation of the closed-loop. We can start by connecting the inputs u from Equation 2-6 and Equation 3-1:

$$\begin{bmatrix} \dot{x} \\ \dot{x}_w \\ \dot{x}_z \\ \dot{\tilde{\mu}}_x \\ \dot{\tilde{\mu}}_u \\ \dot{u} \end{bmatrix} = \begin{bmatrix} A & C_w & 0 & 0 & 0 & B \\ 0 & A_w & 0 & 0 & 0 & 0 \\ 0 & 0 & A_z & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathcal{D} - \kappa_x \tilde{C}^\top \tilde{\Pi}_z \tilde{C} - \kappa_x (\mathcal{D} - \tilde{A})^\top \tilde{\Pi}_w (\mathcal{D} - \tilde{A}) & -\kappa_x \tilde{C}^\top \tilde{\Pi}_z \tilde{D} + \kappa_x (\mathcal{D} - \tilde{A})^\top \tilde{\Pi}_w \tilde{B} & 0 \\ 0 & 0 & 0 & -\kappa_u \tilde{D}^\top \tilde{\Pi}_z \tilde{C} + \kappa_u \tilde{B}^\top \tilde{\Pi}_w (\mathcal{D} - \tilde{A}) & \mathcal{D} - \kappa_u \tilde{D}^\top \tilde{\Pi}_z \tilde{D} - \kappa_u \tilde{B}^\top \tilde{\Pi}_w \tilde{B} - \kappa_u \tilde{\Pi}_u & 0 \\ 0 & 0 & 0 & \rho \tilde{G}^\top \tilde{\Pi}_z \tilde{C} & \rho \tilde{G}^\top \tilde{\Pi}_z \tilde{D} & 0 \end{bmatrix} \begin{bmatrix} x \\ x_w \\ x_z \\ \tilde{\mu}_x \\ \tilde{\mu}_u \\ u \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & B_w & 0 \\ 0 & 0 & 0 & B_z \\ \kappa_x \tilde{C}^\top \tilde{\Pi}_z & 0 & 0 & 0 \\ \kappa_u \tilde{D}^\top \tilde{\Pi}_z & \kappa_u \tilde{\Pi}_u & 0 & 0 \\ -\rho \tilde{G}^\top \tilde{\Pi}_z & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{y} \\ \tilde{\eta} \\ \omega \\ \zeta \end{bmatrix},$$

$$y = \begin{bmatrix} C & 0 & C_z & 0 & 0 & D \end{bmatrix} \begin{bmatrix} x \\ x_w \\ x_z \\ \tilde{\mu}_x \\ \tilde{\mu}_u \\ u \end{bmatrix}.$$

We write this in short as $\dot{x}_{cl} = A_e x_{cl} + B_e u_e$ and $y = C_e x_{cl}$. To make the final step in obtaining the closed-loop, we need to take derivatives of y:

$$\begin{aligned} y &= C_e x_{cl}, \\ \dot{y} &= C_e \dot{x}_{cl} = C_e A_e x_{cl} + C_e B_e u_e \\ &= \begin{bmatrix} CA & 0 & C_z A_z & 0 & 0 & CB \end{bmatrix} x_{cl} + \begin{bmatrix} -D\rho \tilde{G}^\top \tilde{\Pi}_z & 0 & 0 & C_z B_z \end{bmatrix} u_e \\ &= \begin{bmatrix} CA & 0 & C_z A_z & 0 & 0 & CB \end{bmatrix} x_{cl} - D\rho \tilde{G}^\top \tilde{\Pi}_z \tilde{y}, \end{aligned}$$

where we have made use of the fact that $C_z B_z = 0$ if the noise filter is of appropriate order for the AI controller to have that amount of orders p. As we can see, the derivatives of y

depend on $\tilde{y}$ themselves, which is inconvenient. Due to the structure of $\tilde{\Pi}_z$ (as explained in the following section), we can write that $\tilde{G}^\top\tilde{\Pi}_z\tilde{y} = G^\top\Pi_z y$ if $p = 1$. This leads to

$$\begin{aligned}
\dot{y} &= \begin{bmatrix} CA & 0 & C_zA_z & 0 & 0 & CB \end{bmatrix} x_{cl} - D\rho\tilde{G}^\top\tilde{\Pi}_z\tilde{y} \\
&= \begin{bmatrix} CA & 0 & C_zA_z & 0 & 0 & CB \end{bmatrix} x_{cl} - D\rho G^\top\Pi_z y \\
&= \begin{bmatrix} CA & 0 & C_zA_z & 0 & 0 & CB \end{bmatrix} x_{cl} - D\rho G^\top\Pi_z C_e x_{cl} \\
&= \begin{bmatrix} CA - D\rho G^\top\Pi_z C & 0 & C_zA_z - D\rho G^\top\Pi_z C_z & 0 & 0 & CB - D\rho G^\top\Pi_z D \end{bmatrix} x_{cl}.
\end{aligned}$$

As can be seen, if $D \neq 0$, the learning rates will appear quadratically in the closed-loop system matrix due to them appearing in the first column of B_e and in the equation for $\dot{y}$. For $D = 0$ and $p = 1$, we have that

$$\begin{bmatrix} \dot{x} \\ \dot{x}_w \\ \dot{x}_z \\ \dot{\tilde{\mu}}_x \\ \dot{\tilde{\mu}}_u \\ \dot{u} \end{bmatrix} = \begin{bmatrix}
A & C_w & 0 & 0 & 0 & B \\
0 & A_w & 0 & 0 & 0 & 0 \\
0 & 0 & A_z & 0 & 0 & 0 \\
\kappa_x\tilde{C}^\top\tilde{\Pi}_z\begin{bmatrix} C \\ CA \end{bmatrix} & 0 & \kappa_x\tilde{C}^\top\tilde{\Pi}_z\begin{bmatrix} C_z \\ C_zA_z \end{bmatrix} & \mathcal{D} - \kappa_x\tilde{C}^\top\tilde{\Pi}_z\tilde{C} - \kappa_x(\mathcal{D} - \tilde{A})^\top\tilde{\Pi}_w(\mathcal{D} - \tilde{A}) & \kappa_x(\mathcal{D} - \tilde{A})^\top\tilde{\Pi}_w\tilde{B} & \kappa_x\tilde{C}^\top\tilde{\Pi}_z\begin{bmatrix} 0 \\ CB \end{bmatrix} \\
0 & 0 & 0 & \kappa_u\tilde{B}^\top\tilde{\Pi}_w(\mathcal{D} - \tilde{A}) & \mathcal{D} - \kappa_u\tilde{B}^\top\tilde{\Pi}_w\tilde{B} - \kappa_u\tilde{\Pi}_u & 0 \\
-\rho\tilde{G}^\top\tilde{\Pi}_z\begin{bmatrix} C \\ CA \end{bmatrix} & 0 & -\rho\tilde{G}^\top\tilde{\Pi}_z\begin{bmatrix} C_z \\ C_zA_z \end{bmatrix} & \rho\tilde{G}^\top\tilde{\Pi}_z\tilde{C} & 0 & -\rho\tilde{G}^\top\tilde{\Pi}_z\begin{bmatrix} 0 \\ CB \end{bmatrix}
\end{bmatrix} \begin{bmatrix} x \\ x_w \\ x_z \\ \tilde{\mu}_x \\ \tilde{\mu}_u \\ u \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 \\ 0 & B_w & 0 \\ 0 & 0 & B_z \\ 0 & 0 & 0 \\ \kappa_u\tilde{\Pi}_u & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{\eta} \\ \omega \\ \zeta \end{bmatrix},$$

$$y = \begin{bmatrix} C & 0 & HC_z & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ x_w \\ x_z \\ \tilde{\mu}_x \\ \tilde{\mu}_u \\ u \end{bmatrix}.$$

The only tuning parameters that appear non-linearly in the closed-loop system matrix are the input learning rate and input precision, in the term $\kappa_u\tilde{\Pi}_u$. This can be easily resolved by doing a change of coordinates $K_u = \kappa_u\tilde{\Pi}_u$ and obtaining the original precision matrix by $\tilde{\Pi}_u = \kappa_u^{-1}K_u$, making sure that κ_u is non-singular.

For higher generalised orders ($p > 1$), we can quickly verify that even if $D = 0$, the learning rates will appear non-linearly in the closed-loop system matrix due to the fact that

$$\begin{aligned}\ddot{y} &= \begin{bmatrix} CA & 0 & C_z A_z & 0 & 0 & CB \end{bmatrix} \dot{x}_{cl} \\ &= \begin{bmatrix} CA & 0 & C_z A_z & 0 & 0 & CB \end{bmatrix} A_e x_{cl} + \begin{bmatrix} CA & 0 & C_z A_z & 0 & 0 & CB \end{bmatrix} B_e u_e\end{aligned}$$

depends on the learning rate ρ, which will get multiplied with the first column in B_e, leading again to quadratic terms.

This answers the research question. Only for $p = 1$ and $D = 0$, the learning rates appear linearly in the closed-loop matrix. This means that for the second comparison, where the orders will be higher, we will have to make use of tuning techniques that no longer have good guarantees on optimality or convergence.

3-3 Precision matrix

The precision matrices have not yet been defined. They denote the inverse of the variances of the Gaussian distribution that is assumed for the noises, which are given by

$$\Sigma = \begin{bmatrix} \mathbb{E}[w(t)w(t)] & \mathbb{E}[w(t)\dot{w}(t)] & \mathbb{E}[w(t)\ddot{w}(t)] & \mathbb{E}[w(t)w^{(3)}(t)] & \cdots \\ \mathbb{E}[\dot{w}(t)w(t)] & \mathbb{E}[\dot{w}(t)\dot{w}(t)] & \mathbb{E}[\dot{w}(t)\ddot{w}(t)] & \mathbb{E}[\dot{w}(t)w^{(3)}(t)] & \cdots \\ \mathbb{E}[\ddot{w}(t)w(t)] & \mathbb{E}[\ddot{w}(t)\dot{w}(t)] & \mathbb{E}[\ddot{w}(t)\ddot{w}(t)] & \mathbb{E}[\ddot{w}(t)w^{(3)}(t)] & \cdots \\ \mathbb{E}[w^{(3)}(t)w(t)] & \mathbb{E}[w^{(3)}(t)\dot{w}(t)] & \mathbb{E}[w^{(3)}(t)\ddot{w}(t)] & \mathbb{E}[w^{(3)}(t)w^{(3)}(t)] & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

There are multiple ways to calculate this matrix. First of all, a way is shown to calculate this using the auto-correlation of the noise, which is the method that Friston also uses [17]. Secondly, one can calculate the matrix from the impulse response of the filter that is used to convolute WGN. These methods can both be used to calculate the precision matrix for the extended system in Equation 2-2, used in the second comparison. The input precision matrix depends on the precision of the priors, which is more of a trade-off between estimation of the inputs and setting the estimates of the inputs equal to the priors. This matrix can be chosen freely by the designer. A MATLAB implementation to calculate the precision matrix for different filters is provided in section A-3. Lastly, we provide a short discussion on the precision matrix for WGN, which is needed in the first comparison.

3-3-1 From auto-correlation

We have that

$$\Sigma = \sigma^2 \begin{bmatrix} 1 & 0 & \ddot{\rho}(0) & 0 & \cdots \\ 0 & -\ddot{\rho}(0) & 0 & -\rho^{(4)}(0) & \cdots \\ \ddot{\rho}(0) & 0 & \rho^{(4)}(0) & 0 & \cdots \\ 0 & -\rho^{(4)}(0) & 0 & -\rho^{(6)}(0) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3-2}$$

for stationary processes, the derivation for which has been shown in [10, Ch. 7.4]. $\gamma(0) = \sigma^2$ and $\rho(t) = \frac{\gamma(t)}{\sigma^2}$, which is the normalized auto-correlation. To compute more than one order of the precision matrix, $\rho(t)$ thus has to be differentiable multiple times at 0. Note that the one-order-difference cross-terms $\mathbb{E}[w(t)\dot{w}(t)]$, and similarly $\mathbb{E}[\dot{w}(t)\ddot{w}(t)]$, etc., are always 0.

The auto-correlation of a Wide-Sense Stationary (WSS) process with auto-correlation $\gamma_x(t)$ passed trough a stable LTI filter $h(t)$ is given by the convolution $\gamma_y(t) = h * \hat{h} * \gamma_x(t)$, with $\hat{h}(t) = h^*(-t)$, where * denotes the complex conjugate [19, Ch. 8.1]. For a (zero-mean, normalized) WGN process, $\gamma_x(t) = \delta(t)$, where $\delta(t)$ is the Dirac delta function [19, p.251]. So, if we want to calculate the auto-correlation of WGN being passed trough a stable LTI filter h, we can simply calculate $\gamma(t) = h * \hat{h}(t)$. This can also be done in the spectral domain: the Power Spectral Density (PSD) $S(j\omega)$ of a WSS signal is the Fourier transform of the auto-correlation [19, Ch. 8.2], so we can simply calculate

$$S(j\omega) = |H(j\omega)|^2 = H(j\omega)H(-j\omega).$$

Using these relations, we can calculate the precision matrix.

Example Let's say that we have WGN, that is fed through the causal and stable LTI filter with transfer function $H(s) = \frac{1}{(s+1)^2}$, or in time domain, is convoluted with $h(t) = te^{-t}u(t)$, with $u(t)$ the Heaviside step function. The spectral density is given by

$$\begin{aligned} S(j\omega) &= H(j\omega)H(-j\omega) \\ &= \frac{1}{(1+j\omega)^2}\frac{1}{(1-j\omega)^2} \\ &= \frac{1}{(1+\omega^2)^2}. \end{aligned}$$

This leads to the auto-correlation $\gamma(t) = \frac{1}{4}\left(|t|e^{-|t|} + e^{-|t|}\right)$, with $\sigma^2 = \frac{1}{4}$ and normalized auto-correlation $\rho(t) = |t|e^{-|t|} + e^{-|t|}$. The derivatives of this are:

$$\begin{aligned} \frac{d\rho(t)}{dt} = \dot{\rho}(t) &= -te^{-|t|}, \\ \ddot{\rho}(t) &= |t|e^{-|t|} - e^{-|t|}, \\ \rho^{(3)}(t) &= -\frac{e^{-|x|}x(-2+|x|)}{|x|}. \end{aligned}$$

The third derivative does not have a value at 0. The precision matrix will thus be of size 2 and become:

$$\begin{aligned} \Pi(\rho) &= \sigma^{-2}\begin{bmatrix} 1 & 0 \\ 0 & -\ddot{\rho}(0) \end{bmatrix}^{-1} \\ &= \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}. \end{aligned}$$

3-3-2 II: From impulse response

To calculate the covariance $\mathbb{E}[w(t)z(t)]$ of two signals $w(t)$ and $z(t)$, which are obtained by filtering WGN with filters with real impulse responses h_w and h_z, we can just calculate the integral $\int_{-\infty}^{\infty} h_w(\tau)h_z(\tau)d\tau$. This also extends to the derivatives, and follows simply from the derivations in [19, Ch. 8.1].

Example Let us take the same example as in the previous section. We can simply directly calculate the terms that are necessary:

$$\begin{aligned}
\mathbb{E}[w(t)w(t)] &= \int_{-\infty}^{\infty} h(\tau)h(\tau)d\tau \\
&= \int_{-\infty}^{\infty} \tau e^{-\tau}u(\tau)\tau e^{-\tau}u(\tau)d\tau \\
&= \int_{0}^{\infty} \tau^2 e^{-2\tau}d\tau \\
&= \frac{1}{4}, \\
\mathbb{E}[w(t)\dot{w}(t)] &= \int_{-\infty}^{\infty} h(\tau)\dot{h}(\tau)d\tau \\
&= [h(\tau)h(\tau)]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} \dot{h}(\tau)h(\tau)d\tau \text{ (integration by parts)} \\
&= \frac{1}{2}\left[h(\tau)h(\tau)\right]_{-\infty}^{\infty} \\
&= \frac{1}{2}\left[\tau^2 e^{-2\tau}u^2(\tau)\right]_{-\infty}^{\infty} \\
&= 0, \\
\mathbb{E}[\dot{w}(t)\dot{w}(t)] &= \int_{-\infty}^{\infty} \dot{h}(\tau)\dot{h}(\tau)d\tau \\
&= \int_{-\infty}^{\infty} \left(e^{-\tau}u(\tau) - \tau e^{-\tau}u(\tau)\right)\left(e^{-\tau}u(\tau) - \tau e^{-\tau}u(\tau)\right)d\tau \\
&= \int_{0}^{\infty} e^{-2\tau}d\tau + \int_{0}^{\infty} -2\tau e^{-2\tau}d\tau + \int_{0}^{\infty} \tau^2 e^{-2\tau}d\tau \\
&= \frac{1}{2} - \frac{1}{2} + \frac{1}{4} = \frac{1}{4}.
\end{aligned}$$

Filling these terms in will result in the same precision matrix as derived earlier.

3-3-3 I: For White Gaussian Noise

In continuous time, the precision matrix of WGN is not defined very well. White noise formally has a covariance of $\sigma^2\delta(0)$ [19, p.251], which means that for a filter which just passes the white noise through, the precision matrix is 0, leading to a trivial FE. In a Kalman filter, which assumes WGN, the matrices that are commonly called the 'covariance' matrices are actually the matrices containing the PSD of the noise signal.

When introducing the precision matrix, Friston et al. decompose the matrix in two parts: a temporal and a stationary part. This corresponds to the matrix containing the normalized autocorrelation and its derivatives and the σ^2 in Equation 3-2, respectively. So in the reduction to $p = 0$ when we have WGN, we should drop the temporal part of the covariance. Friston et al. also omit this part when comparing Dynamic Expectation Maximization (DEM) and Kalman filtering by assuming Gaussian noise with infinite roughness [17].

Chapter 4

Comparison of LQG control and AI

In this chapter, Linear-Quadratic Gaussian (LQG) control is compared to Active Inference (AI). First, a high-level comparison is done between the two state-space representations to answer the first research question. After that, the White Gaussian Noise (WGN) setting with known inputs is considered for both controllers, and the theoretical equivalence between the two is shown by choosing the tuning parameters in a certain way, answering the second question. Finally, a practical comparison is done by simulating systems with colored noise, where the noise is formed by convoluting WGN with a Linear Time-Invariant (LTI) filter, giving an answer to the last research question.

4-1 High-level comparison between the two controllers

In this section, we will answer the research question:

> *What are equivalences and differences in the state-space equations of the two controllers?*

Looking at both controllers in Equation 2-5 and Equation 3-1, we can immediately see a few differences. The AI controller has $(p+1)(n+m)+m$ states, whereas the LQG controller only has n states. Furthermore, AI does integral control, whereas the LQG controller is a static feedback gain on the state estimates. A similarity in the controllers is that both do some kind of filtering of the system states.

Even when $p=0$ and we assume that we know the inputs, which is more similar to the LQG setting, the AI controller has m states extra, corresponding to the integral control. One can hypothesize that the LQG control feedback gain is some sort of steady-state solution to a differential equation for the control input. So, if the two are to be equivalent, at the very least inputs have to be known, p has to be 0 and we need to find the static solution to the integral controller. In the next section, we show how this can be done.

4-2 Comparison I: Equivalence for WGN with knowledge of inputs

In this section, we will answer the research question:

When are the two controllers equivalent?

Following the reasoning in the previous section, we start with AI where inputs are known. To go to the $p = 0$ (WGN) case, we begin with the Free Energy (FE) for only one generalised coordinate, which is given by

$$\begin{aligned} F(\tilde{\mu}_x, y, \eta, u) = &\frac{1}{2}(y - g(\mu_x, u))^\top \Pi_z (y - g(\mu_x, u)) + \frac{1}{2}(\mu_{x'} - f(\mu_x, u))^\top \Pi_w (\mu_{x'} - f(\mu_x, u)) \\ &+ \frac{1}{2}(u - \eta)^\top \Pi_u (u - \eta), \end{aligned}$$

where the generative model is given by $g(\mu_x, u) = C\mu_x + Du$ and $f(\mu_x, u) = A\mu_x + Bu$. For this FE, both controllers from subsection 3-1-3 are reduced to:

$$\begin{aligned} &\begin{bmatrix} \dot{\mu}_x \\ \dot{\mu}_{x'} \\ \dot{u} \end{bmatrix} = \\ &\begin{bmatrix} -\kappa_x(C^\top \Pi_z C + A^\top \Pi_w A) & I + \kappa_x A^\top \Pi_w & -\kappa_x(C^\top \Pi_z D + A^\top \Pi_w B) \\ \kappa_{x'} \Pi_w A & -\kappa_{x'} \Pi_w & \kappa_{x'} \Pi_w B \\ -\rho((C\hat{A}^{-1}B)^\top \Pi_z C + B^\top \Pi_w A) & \rho B^\top \Pi_w & -\rho((C\hat{A}^{-1}B)^\top \Pi_z D + B^\top \Pi_w B + \Pi_u) \end{bmatrix} \begin{bmatrix} \mu_x \\ \mu_{x'} \\ u \end{bmatrix} \\ &\quad + \begin{bmatrix} \kappa_x C^\top \Pi_z & 0 \\ 0 & 0 \\ \rho(C\hat{A}^{-1}B)^\top \Pi_z & \rho \Pi_u \end{bmatrix} \begin{bmatrix} y \\ \eta \end{bmatrix}, \\ &\quad u = \begin{bmatrix} 0 & 0 & I \end{bmatrix} \begin{bmatrix} \mu_x \\ \mu_{x'} \\ u \end{bmatrix}. \end{aligned}$$

This reduction is similar to the reductions found in [18] and [4], where with $p = 0$ only the first row of the generalised equations of motion $\mathcal{D}\tilde{x}_x = f(\tilde{x}, u) + \tilde{w}$ is taken ($x' = Ax + Bu + w$). In [2], the reduction is done slightly differently, as there for $p = 0$ only the first order of $\tilde{x}$ is kept ($0 = Ax + Bu + w$). Note that we take the inverse Power Spectral Density (PSD) for Π_w and Π_z if we have WGN, instead of the inverse covariances, as explained in subsection 3-3-3.

As can be seen, the estimate of the first derivative (the second row in the state-space update equations) does not depend on any inputs, and we can set the learning rate very high with respect to the other updates, which leads to the static solution $\mu_{x'} = A\mu_x + Bu$, which we can fill in to obtain

$$\begin{aligned} &\begin{bmatrix} \dot{\mu}_x \\ \dot{u} \end{bmatrix} = \begin{bmatrix} A - \kappa_x C^\top \Pi_z C & B - \kappa_x C^\top \Pi_z D \\ -\rho(C\hat{A}^{-1}B)^\top \Pi_z C & -\rho((C\hat{A}^{-1}B)^\top \Pi_z D + \Pi_u) \end{bmatrix} \begin{bmatrix} \mu_x \\ u \end{bmatrix} + \begin{bmatrix} \kappa_x C^\top \Pi_z & 0 \\ \rho(C\hat{A}^{-1}B)^\top \Pi_z & \rho \Pi_u \end{bmatrix} \begin{bmatrix} y \\ \eta \end{bmatrix}, \\ &\quad u = \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} \mu_x \\ u \end{bmatrix}. \end{aligned}$$

This is AI with $p = 0$. If we neglect the controller part, we can immediately see that this is equivalent to the Kalman filter in Equation 2-4 for $\kappa_x = P$. We would now like to use the same trick to eliminate the extra controller states. The controller updates, however, depend on the outputs y, which will lead to the static control action depending on y, which is not the case for the Linear-Quadratic Regulator (LQR) in Equation 2-5. Thus, we assume that the outputs are in expectation given by $y = Gu = -C\hat{A}^{-1}Bu + Du$. Doing this, we can write the second row of the controller as

$$\dot{u} = -\rho(C\hat{A}^{-1}B)^\top \Pi_z C\mu_x - \rho((C\hat{A}^{-1}B)^\top \Pi_z C\hat{A}^{-1}B + \Pi_u)u + \rho\Pi_u\eta.$$

If we set the controller gain very high, just like we did earlier for the estimation gain for $\mu_{x'}$, we arrive at the solution to $\dot{u} = 0$:

$$u = ((C\hat{A}^{-1}B)^\top \Pi_z C\hat{A}^{-1}B + \Pi_u)^{-1}(-(C\hat{A}^{-1}B)^\top \Pi_z C\mu_x + \Pi_u\eta),$$

which gives the final controller equation:

$$\begin{aligned}\dot{\mu}_x &= (A - LC - BK + LDK)\mu_x + \begin{bmatrix} L & BR^{-1}\Pi_u - LDR^{-1}\Pi_u \end{bmatrix} \begin{bmatrix} y \\ \eta \end{bmatrix}, \\ u &= -K\mu_x + \begin{bmatrix} 0 & R^{-1}\Pi_u \end{bmatrix} \begin{bmatrix} y \\ \eta \end{bmatrix},\end{aligned}$$

with $L = \kappa_x C^\top \Pi_z$, $K = R^{-1}B^\top (C\hat{A}^{-1})^\top \Pi_z C$ and $R = (C\hat{A}^{-1}B)^\top \Pi_z C\hat{A}^{-1}B + \Pi_u$. The two reductions by setting a learning rate very high, essentially entail directly going to the minimum of the FE function, instead of doing a descent. Not doing this for μ_x is similar to a state observer: if a state observer gain is set very high, we only consider the most recent measurements and do not take into account the model estimates and predictions. In Table 4-1, the terms of the AI controller are compared with the nominal LQG controller from Equation 2-5.

Table 4-1: A comparison of the terms of the reduced AI controller and the nominal LQG controller. Differences are highlighted in bold.

	AI	LQG
A_c	$A - LC - BK + LDK$	$A - LC - BK + LDK$
B_c	$\begin{bmatrix} L & B\mathbf{R^{-1}\Pi_u} - LD\mathbf{R^{-1}\Pi_u} \end{bmatrix}$	$\begin{bmatrix} L & B\mathbf{KB} - LD\mathbf{KB} \end{bmatrix}$
C_c	$-K$	$-K$
D_c	$\begin{bmatrix} 0 & \mathbf{R^{-1}\Pi_u} \end{bmatrix}$	$\begin{bmatrix} 0 & \mathbf{KB} \end{bmatrix}$
L	$\boldsymbol{\kappa_x} C^\top \Pi_z$	$\mathbf{P}C^\top \Pi_z$
K	$R^{-1}B^\top(\mathbf{C\hat{A}^{-1}})^\top \mathbf{\Pi_z C}$	$R^{-1}B^\top \mathbf{S}$
R	$(\mathbf{C\hat{A}^{-1}B})^\top \mathbf{\Pi_z C\hat{A}^{-1}B} + \mathbf{\Pi_u}$	**Given**

Hence, the two controllers are equivalent if we set the estimation learning rate $\kappa_x = P$, the solution to the estimation Algebraic Riccati Equation (ARE), assume that the solution to the control ARE is given by $S = (C\hat{A}^{-1})^\top \Pi_z C$, which is not necessarily symmetric or positive definite, and set the input precision matrix $\Pi_u = B^\top SB$.

Filling the obtained S and R into the controller ARE we obtain:

$$\begin{aligned}
0 =& A^\top (C\hat{A}^{-1})^\top \Pi_z C + ((C\hat{A}^{-1})^\top \Pi_z C)^\top A + Q \\
&- ((C\hat{A}^{-1})^\top \Pi_z C)^\top B((C\hat{A}^{-1}B)^\top \Pi_z C\hat{A}^{-1}B + B^\top (C\hat{A}^{-1})^\top \Pi_z CB)^{-1} B^\top (C\hat{A}^{-1})^\top \Pi_z C \\
Q =& - A^\top \hat{A}^{-\top} C^\top \Pi_z C - C^\top \Pi_z C\hat{A}^{-1} A \\
&+ C^\top \Pi_z C\hat{A}^{-1} B((C\hat{A}^{-1}B)^\top \Pi_z C(\hat{A}^{-1} + I)B)^{-1} (C\hat{A}^{-1}B)^\top \Pi_z C
\end{aligned}$$

In Table 4-2, the different tuning parameters for equivalence of AI and LQG are summarized as a function of $\hat{A}$. Note that the LQG terms Q, R and S are not necessarily positive (semi-)definite, nor that S is guaranteed to be symmetric. This is also the case for the precision matrix of the inputs Π_u.

Table 4-2: Comparison of the tuning parameters of AI and LQG for equivalence as a function of $\hat{A}$. S is included for ease of comparison, as it actually not a tuning parameter but dependent on Q and R.

AI	LQG
$\kappa_x = P$	$Q = C^\top \Pi_z C\hat{A}^{-1} B((C\hat{A}^{-1}B)^\top \Pi_z C(\hat{A}^{-1} + I)B)^{-1} (C\hat{A}^{-1}B)^\top \Pi_z C$
$\kappa_{x'} \to \infty$	$-A^\top \hat{A}^{-\top} C^\top \Pi_z C - C^\top \Pi_z C\hat{A}^{-1} A$
$\rho \to \infty$	$R = (C\hat{A}^{-1}B)^\top \Pi_z C(\hat{A}^{-1} + I)B$
$\Pi_u = (C\hat{A}^{-1}B)^\top \Pi_z CB$	$S = \hat{A}^{-\top} C^\top \Pi_z C$

The closed-loop system matrix of this reduced AI on the system from Equation 2-1 is now given by

$$A_{cl} = A - B((C\hat{A}^{-1}B)^\top \Pi_z C(\hat{A}^{-1} + I)B)^{-1} (C\hat{A}^{-1}B)^\top \Pi_z C.$$

We thus have to choose the $\hat{A}$ matrix right to get a stabilizing controller, which is in contrast to LQG, which guarantees a stabilizing controller for any $Q \geq 0, R > 0$ on any linear system.

4-3 Comparison II: Filtered WGN without knowledge of inputs

In this section, we will answer the research question:

How does tuned AI compare to LQG control in terms of LQG costs?

We will do this by means of a numerical comparison between the extended LQG from subsection 2-2-2 and the original AI controller from Equation 3-1. As explained in section 4-1, there are now inherent differences between the two controllers. Both controllers are run on the extended system from Equation 2-2. For the forward model for the AI controller, we simply set $\hat{A} = A$, letting the rest of the tuning parameters stabilize the closed-loop. The main systems are single-input, single-output (SISO) or multiple-input, multiple-output (MIMO) systems of increasing orders n, with the noise filters having a relative order of 3 or 6, such that $p = 2$ or $p = 5$, respectively. The filters have been scaled such that the precision at the first order is 1. The reference or prior $\tilde{\eta}$ was set to 0.

Tuning of the AI controller is done to minimize the same cost (Equation 2-3) as LQG does, to ensure a fair comparison. The cost matrices are set to identity and the final obtained costs are scaled such that the costs obtained by LQG are 1. Because the tuning problem is not easily solved, as elaborated in section 3-2, we make use of the `systune` option in MATLAB, using 20 random starts. For $\tilde{\Pi}_u$, only the elements on the main diagonal are set to be tunable, with values greater than 0. Knowledge of the underlying noise systems is included during the tuning phase. For the derivatives in AI, a high-pass filter was used as approximation.

The results are generated by running the controllers on 100 randomly generated extended systems for one million time steps with MATLAB's `lsim` command. The random noise systems are generated by the code in section A-4; the code for a single realization can be found in section A-5.

In Figure 4-1, a comparison is provided for SISO systems, with $p = 2$. As can be seen, we are able to tune AI very close to the optimal costs that LQG obtained for small system orders. For all of the six situations, there was at least one AI controller that achieved the optimal solution within less than one percent, showing the potential of using more random starts. Only for $n \leq 3$, the medians were within 5%.

In Figure 4-2, a comparison is provided for MIMO systems, with $p = 2$. As is clear from the incurred costs, it gets harder to tune the AI controller such that it behaves optimally, especially for higher orders. Now, for $n \leq 3$, the medians were within 100% of the optimal solutions. Lastly, in Figure 4-3, a comparison is provided for SISO systems with $p = 5$. Similarly for the MIMO case, it is hard to obtain optimal behaviour and now even the medians are infinite, corresponding to failed tuning (unstable solution).

It thus seems that for higher-dimensional problems, i.e. n, p or m or q becomes higher, it becomes increasingly hard to obtain a close-to-optimal controller. But this is of course to be expected: the dimensionality of the optimization problem increases, while we are dealing with a non-linear optimization problem. Of course, we could have tried more random starts and might have ended up with better controllers, but this is not certain as of yet. For smaller problems, we are able to tune AI such that its behaviour is very close to the optimal LQG solution.

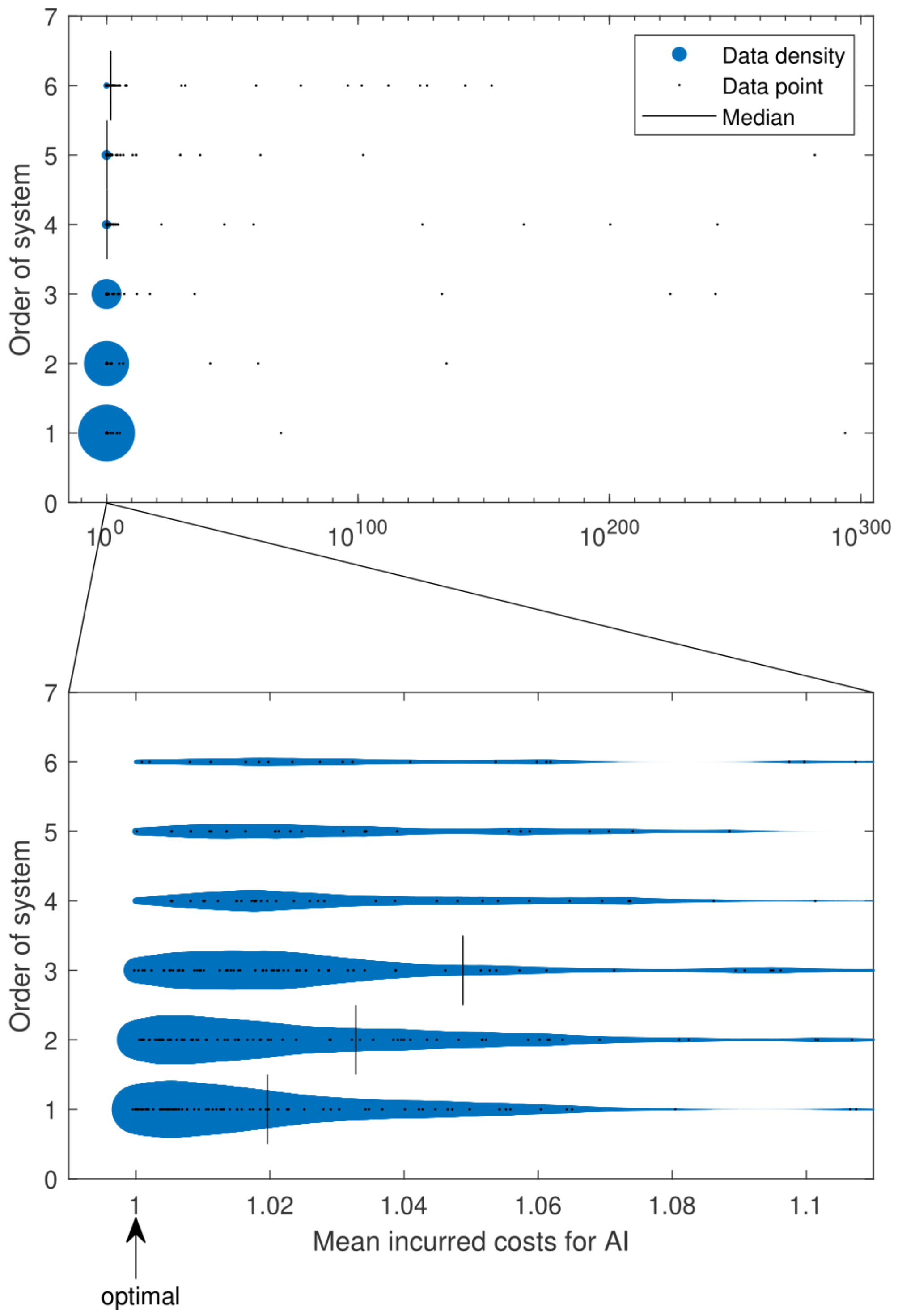


Figure 4-1: A comparison of AI and LQG on the extended system from Equation 2-2. The base system is a SISO system of varying order n. The noise systems have a relative order of 3, so $p = 2$. The top figure shows a very zoomed-out plot of the incurred costs of AI compared to LQG. The bottom figure shows the incurred AI costs within 11% of the optimal (LQG) costs.

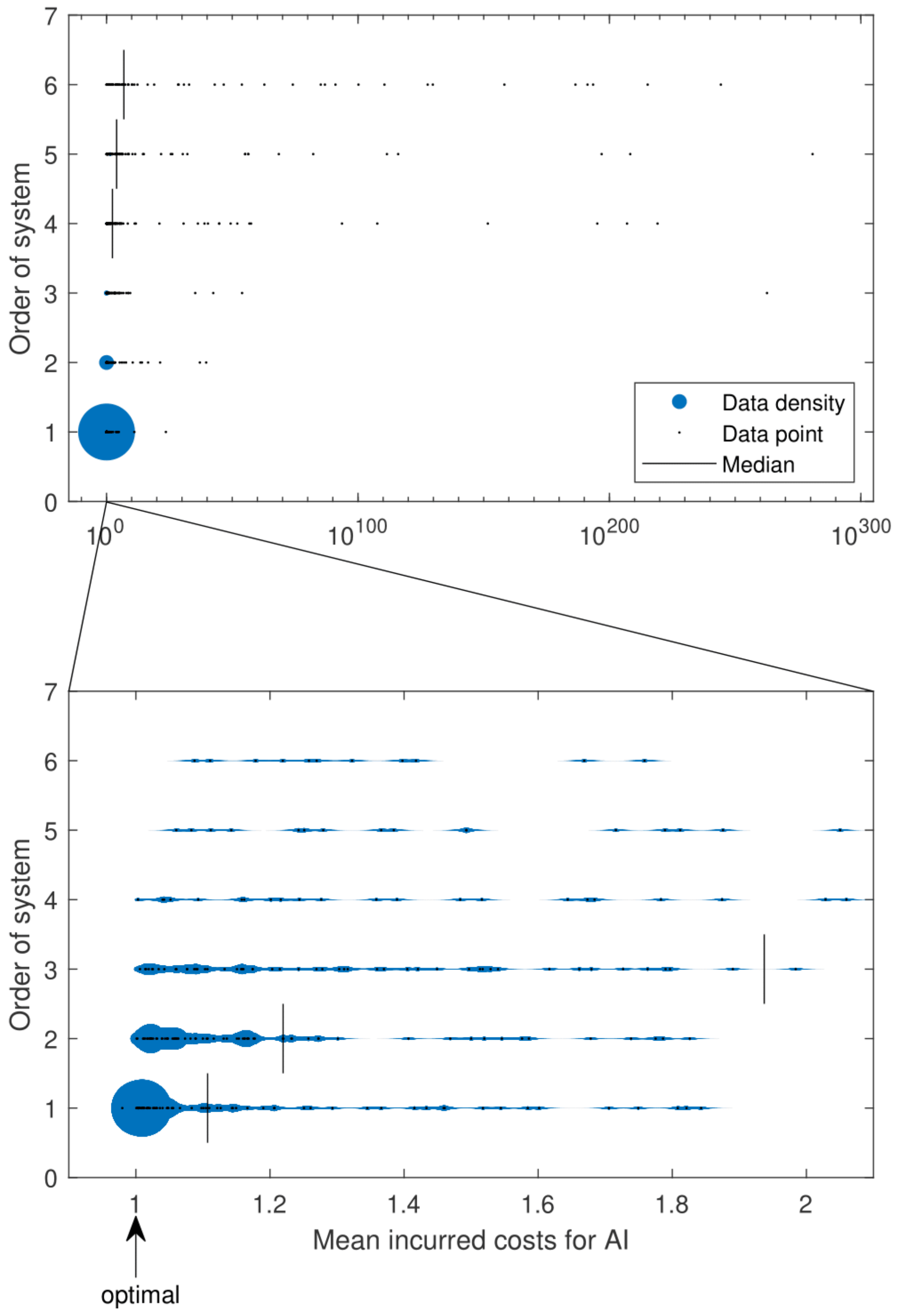


Figure 4-2: A comparison of AI and LQG on the extended system from Equation 2-2. The base system is a MIMO system of varying order n, with $m = q = 3$. The noise systems have a relative order of 3, so $p = 2$. The top figure shows a very zoomed-out plot of the incurred costs of AI compared to LQG. The bottom figure shows the incurred AI costs within 110% of the optimal (LQG) costs.

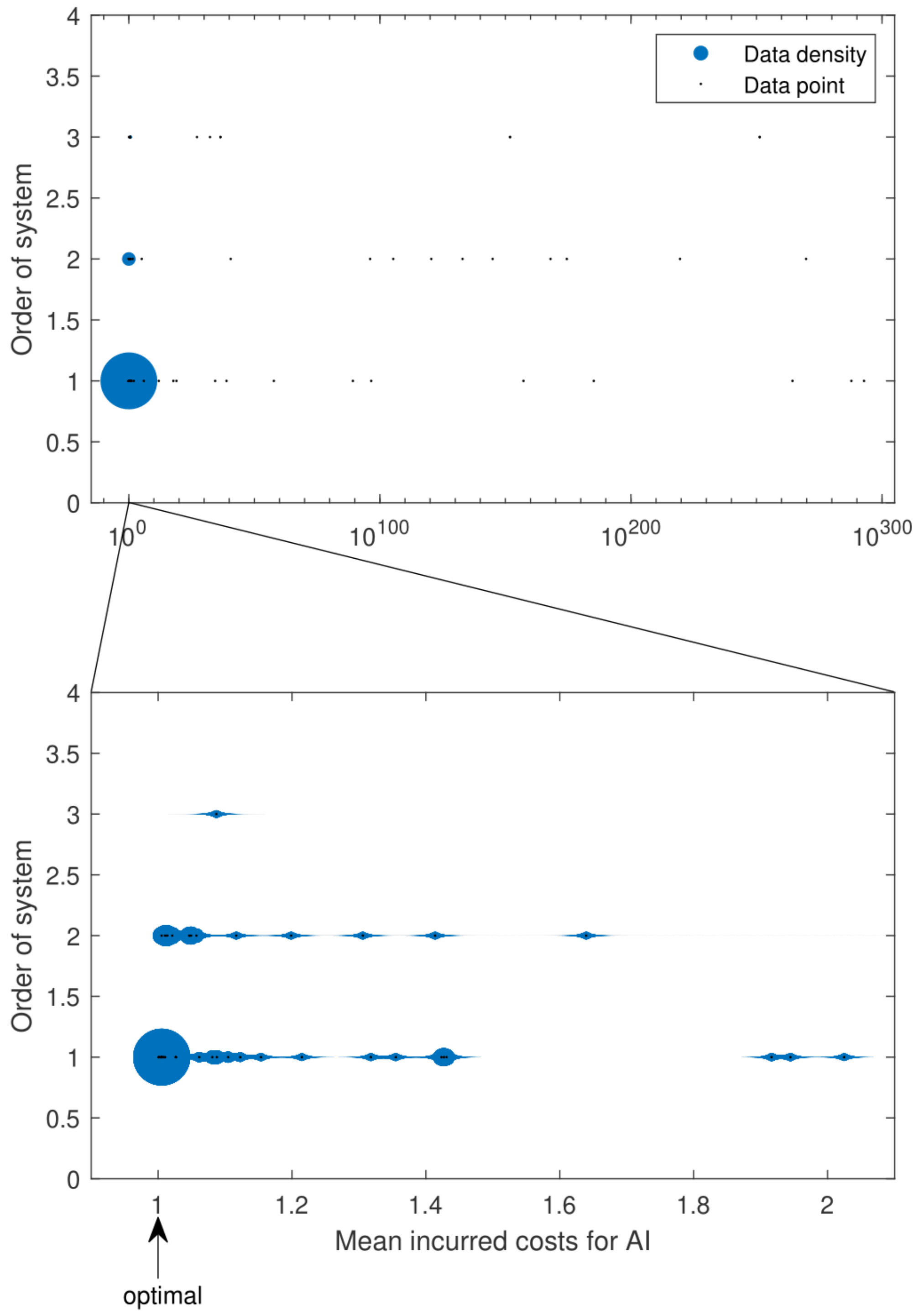


Figure 4-3: A comparison of AI and LQG on the extended system from Equation 2-2. The base system is a SISO system of varying order n. The noise systems have a relative order of 6, so $p = 5$. The top figure shows a very zoomed-out plot of the incurred costs of AI compared to LQG. The bottom figure shows the incurred AI costs within 110% of the optimal (LQG) costs. The median values are not displaced, as they are now all infinite (failed tuning).

Chapter 5

Conclusions and recommendations

In this chapter, we give the conclusions of this thesis work. We also discuss Free Energy (FE) as a Lyapunov function. Lastly, some recommendations for further research are given.

5-1 Conclusions and discussion

In this thesis work, we showed how Active Inference (AI) is formulated for Linear Time-Invariant (LTI) systems. This was done for 3 different formulations of the FE. First, the original implementation of Friston was given. After that, it was shown that this could be reduced to the AI implementation in [18] by simply setting the prior precision very high. Lastly, a new implementation was given, where it was assumed that the controller has access to the inputs. This could be unlike some biological systems, where efferent copies are unavailable, but is very natural for control systems as used by engineers.

As knowledge of inputs is one of the requirements for the Linear-Quadratic Gaussian (LQG) controller, we reduced the known-inputs AI to the White Gaussian Noise (WGN) noise case - the case when LQG is optimal. This reduction was done similarly to some existing literature. After that, it was shown that for particular choices of the learning rates and the prior precision, AI reduces to the same structure as LQG, which is the first time that such a result is shown in the literature. A comparison was made between the learning rates and prior precision for AI on the one hand, and the cost matrices for LQG on the other hand. Doing this, we found that in order to stabilize AI, we should choose the forward model appropriately. This stands in contrast to LQG, where a stabilizing controller is found for any system and any cost function. Furthermore, the matrices that are normally used for LQG no longer adhere to their positive (semi-)definite constraints, making it hard to obtain intuitive relations between choices of cost matrices, prior precision and forward model.

Other than the reduction of AI to the LQG setting, we extended LQG to a setting closer to AI. We obtained the optimal LQG controller for this new setting, and compared how close the original AI implementation came to the behaviour of this optimal controller. It turned out, that for small systems, i.e. number of states, inputs, outputs and generalised states relatively small, we could tune AI very close to optimal behaviour. When the systems grew in size, however, the tuning became harder. The conception is that AI is still able to show close-to-optimal behaviour for these larger systems, if the tuning is done optimally. This has, however, not been proven or shown.

As a point of discussion, in the second comparison we obtained the optimal controller for the LQG cost function (Equation 2-3), which is the LQG controller if the original noise is WGN. So the comparison was more to see whether AI could reach the optimality of LQG. It would be fair to also include a comparison where LQG is not guaranteed to be optimal, for example for noises that cannot be modelled with LTI systems (e.g. Gaussian convolved noise). AI might be better in that case; [2] has already made a start in comparing Dynamic Expectation Maximization (DEM) and Kalman filtering and has found that DEM outperforms Kalman filtering.

On a more fundamental note, Friston has argued that 'under the Free Energy Principle (FEP) (i.e., the brain changes to minimise its FE), the FE becomes a Lyapunov function for the brain' [16]. This is of course the very definition of a Lyapunov function: if a system only moves downward on such a function, where the minimum is at the equilibrium, the equilibrium is stable. But this statement can be mistaken for saying that the FE is a Lyapunov function for running AI on any system. This is not the case, as still the condition needs to hold that AI actually minimizes the FE, which really depends on the system being controlled, the generative model that is assumed and the choice of tuning parameters. In another work, Friston decomposes the flow of a generative model into a divergence- and curl-free part of the FE [14]. The curl-free part is then the dissipative part, which ensures stability (so he chooses a stable generative model). Doing this, he argues that the FE becomes a Lyapunov function. But this rests on the assumption that 'action eliminates (on average) the difference between the actual and predicted flow' [14], in other words: action needs to be such that the controlled system behaves like a stable generative model. But that is of course the whole point of control and, other than the gradient descent on the FE, Friston gives no methods to achieve this condition.

5-2 Recommendations

As no research stands on its own, and there still remain open questions on this subject, we give here some recommendations for further research:

- Relations between reduced AI and LQG control:
  We obtained relations between the choice of forward model and choice of cost matrices. However, these terms are quite advanced. What are their intuitive explanations?

- Tuning of AI:
  From the results on the second comparison, it became clear that the tuning of the AI controller is a non-trivial task. The tuning is not only needed for optimality, but also even for stability. It is therefore essential that more research is conducted into tuning procedures. During the work on this thesis, linear-matrix inequality techniques were considered for tuning, but the problem becomes quickly non-linear for more generalised coordinates. Also, as was clear from the results of the second comparison, the tuning

problem became harder for more complex systems. However, for low-order systems, we were often able to find a close-to-optimal solution for AI. It would be worthwhile to investigate whether these solutions always exist, even for more complex systems.

- AI and Online Optimization (OO):
  AI has some stability issues, that have also been encountered in this thesis work. In a literature survey that was performed earlier by the author, connections between the OO framework and AI were investigated. Both deal with the minimization of a (quadratic) cost function, and do so mostly using a gradient descent. For the OO framework, a performance measure called *regret* is used. From the literature survey, it became clear that minimizing regret does not directly ensure stabilizing dynamics. Recent developments in OO, however, are increasingly in the direction of OO on LTI systems, see for example [9, 29] and related works. The controllers that are obtained using these techniques are stabilizing. To tackle the stability issues of AI, making the connection between AI and these recent developments could be a useful way forward.

- Comparison of AI and LQG:
  In this thesis, we only considered cases where LQG was guaranteed to be optimal. It would be fair if further comparative research could be done into situations closer to real life, where noises are not easily modelled, white or formed linearly.

- AI on Linear Time-Varying (LTV) systems:
  In this thesis work, we have made a comparison between AI and the infinite-horizon time-invariant LQG problem. LQG controllers, however, can also be defined for LTV systems. AI on LTV systems has not yet been investigated; it would be interesting to see whether the same equivalency can be obtained as is this thesis work. Making the step to LTV systems will require a rewriting of generative models to also include the derivatives of the system matrices.

Appendix A

MATLAB code

A-1 Closed-loop LQG

The following is code to calculate the closed-loop system of Linear-Quadratic Gaussian (LQG) control on the systems in Equation 2-2.

```
function [sysclLQG,IP,p] = closedLoopLQG(sys,sysw,sysz,Q,R,N)
%CLOSEDLOOPLQG Closed-loop system of LQG on a ss system
%
%   [sysclLQG,IP,p] = CLOSEDLOOPLQG(sys,sysw,sysz,Q,R,N)
%   gives the closed loop ss system sysclLQG that results from running
    the
%   optimal (LQG) controller on the system sys, with sysw and sysz the
%   noise systems, having normalized WGN as inputs and the noise w and z
    as
%   output. Q, R and N are the normal LQR weighting matrices.
%
%   Inputs:
%   sys:    System to be controlled, given by
%         .
%         x = Ax + Bu + Gw          {State equation}
%         y = Cx + Du + Hz          {Measurements}
%         such that sys = ss(A,[B G 0],C,[D 0 H])
%
%   sysw:     Process noise filter
%   sysz:     Measurement noise filter
%
%   Q:  Weighing matrix for states (x'Qx)
%   R:  Weighing matrix for inputs (u'Ru)
%   N:  Weighing matrix for states/inputs (x'Nu). Set to 0 if omitted.
%
%   Outputs:
%   sysclAI:  Tuned closed-loop of AI on system sys
%   IP:         A boolean returning TRUE if the LQG controller is proper
%   p:          The amount of derivatives that can be taken for each
    output
%
%
%   Jesse Coehoorn
%   MSc Thesis Systems & Control and Mechanical Engineering

% Systems dimensions (n states, m inputs, q outputs)
nw = size(sysw.A,2);
nz = size(sysz.A,2);
mw = size(sysw.B,2);
mz = size(sysz.B,2);
qw = size(sysw.C,1);
qz = size(sysz.C,1);
n = size(sys.A,2);
m = size(sys.B,2)-qw-qz;
q = size(sys.C,1);

% Noisy system
An = [sys.A sys.B(:,m+1:m+qw)*sysw.C zeros(n,nz);
      zeros(nw,n) sysw.A zeros(nw,nz);
      zeros(nz,n) zeros(nz,nw) sysz.A];
Bu = [sys.B(:,1:m); zeros(nw,m); zeros(nz,m)];
Bn = [zeros(n,mw) zeros(n,mz);
      sysw.B zeros(nw,mz);
      zeros(nz,mw) sysz.B];
Cn = [sys.C zeros(q,nw) sys.D(:,m+qw+1:end)*sysz.C];
D = sys.D(:,1:m);

% Determine amount of derivatives that can be taken
p = ones(q,1);
D2 = zeros(q,mw+mz);
for iq = 1:q
    OBBn = Cn*An^(p(iq)-1)*Bn;
    while sum(OBBn(iq,:)) == 0
        p(iq) = p(iq) + 1;
        OBBn = Cn*An^(p(iq)-1)*Bn;
    end
    D2(iq,:) = OBBn(iq,:);
end

% Determine transformation matrices
OB = obsv(An,Cn);
% Determine M
M2 = [];
for iq = 1:q
    M2 = [M2;OB(iq+q*(0:p(iq)-1),:)];
end
nr = size(M2,1);
M = [null(M2)';M2];
% Determine H2
H2 = [];
for iq = 1:q
```

```
    H2 = [H2;OB(iq+q*p(iq),:)];
end
% Determine G
G = [];
for iq = 1:q
    for id = 1:p(iq)
        if id == 1
            G = [G;[D(iq,:) zeros(1,max(p)*m)]];
        else
            CO = [OB(iq+q*(id-2:-1:0),:)*Bu;D(iq,:)]';
            CO = CO(:)';
            G = [G;[CO zeros(1,(max(p)+1-id)*m)]];
        end
    end
end
for iq = 1:q
    CO = [OB(iq+q*(p(iq)-1:-1:0),:)*Bu;D(iq,:)]';
    CO = CO(:)';
    G = [G;[CO zeros(1,(max(p)-p(iq))*m)]];
end

% Determine transformed system
F = M*An/M;
F11 = F(1:end-nr,1:end-nr);
F12 = F(1:end-nr,end-nr+1:end);
Bu12 = M*Bu;
Bu1 = Bu12(1:end-nr,:);
Bn12 = M*Bn;
Bn1 = Bn12(1:end-nr,:);

% Transformation of observations
H34 = H2/M;
H3 = H34(:,1:end-nr);
H4 = H34(:,end-nr+1:end);

% Kalman filter
sysk = ss(F11,[Bu1 F12 Bn1],H3,[zeros(q,m+nr) D2]);
Qn = eye(mw+mz);
[Kest,~,~] = kalman(sysk,Qn,[]);
Kest.y(q+1:end) = ionames('x1',n+nw+nz-nr);
Kest.u(1:m) = ionames('u',m);
Kest.u(m+1:m+nr) = ionames('x2',nr);
Kest.u(m+nr+1:end) = ionames('yk',q);

% LQR
try N; catch N = zeros(n,m); end
Q = blkdiag(Q,zeros(nw+nz));
N = [N;zeros(nw+nz,m)];
[L,~,~] = lqr(An,Bu,Q,R,N);
L(:,end-nz+1:end) = zeros(m,nz);

% Derivatives
s = tf('s');
```

```
% G*utilde
du = [];
for idu = 0:max(p)
    du = [du;s^idu*eye(m)];
end
Gdu = G(1:end-q,:)*du;
Gddu = G(end-q+1:end,:)*du;
Gdu.u = 'u';
Gdu.y = 'Gdu';
Gddu.u = 'u';
Gddu.y = 'Gddu';
% ytilde
dy = [];
for iq = 1:q
    I = zeros(1,q);
    I(iq) = 1;
    for idy = 0:p(iq)-1
        dy = [dy;s^idy*I];
    end
end
dy.u = 'y';
dy.y = 'dy';
ddy = [];
for iq = 1:q
    I = zeros(1,q);
    I(iq) = 1;
    ddy = [ddy;s^p(iq)*I];
end
ddy.u = 'y';
ddy.y = 'ddy';

% x2, z, and y' signals
x2 = sumblk('x2 = dy - Gdu',sum(p));
z = sumblk('z = ddy - Gddu',q);
H4 = ss(H4);
H4.u = ionames('x2',nr);
H4.y = ionames('H4x2',q);
yk = sumblk('yk = z - H4x2',q);

% LQG system
K = connect(Kest,Gdu,Gddu,dy,ddy,x2,z,yk,H4,[{'u'};{'y'}],[{'x1'};{'x2'
    }]);
sysLQG = -L/M*K;
sysLQG.y = 'u';
IP = true;
try sysLQG = ss(sysLQG,'explicit'); catch, IP = false; end

% Closed loop system
sys = ss(sys.A,sys.B,[sys.C;eye(n)],[sys.D;zeros(n,m+qw+qz)]);
sys.y(1:q) = ionames('y',q);
sys.y(q+1:end) = ionames('x',n);
sys.u(1:m) = ionames('u',m);
sys.u(m+1:m+qw) = ionames('w',qw);
sys.u(m+qw+1:end) = ionames('z',qz);
sysw.u = 'omega';
sysw.y = 'w';
sysz.u = 'zeta';
sysz.y = 'z';
sysclLQG = connect(sys,sysw,sysz,sysLQG,[{'omega'};{'zeta'}],[{'y'};{'x'
    };{'u'}]);

% Filter out artifacts due to inverses etc. Otherwise some poles/zeros
    will
% not cancel out for explicit representation
sysclLQG.A(abs(real(sysclLQG.A))<1e-12) = 0;
sysclLQG.B(abs(real(sysclLQG.B))<1e-12) = 0;
sysclLQG.C(abs(real(sysclLQG.C))<1e-12) = 0;
sysclLQG.D(abs(real(sysclLQG.D))<1e-12) = 0;
if IP == false
    sysclLQG = ss(sysclLQG,'explicit');
end

end


function ioname = ionames(ioname,n)

    if n == 1
        ioname = {ioname};
    else
        ioname = cellstr(ioname+"("+(1:n)'+")");
    end
end
```

A-2 Closed-loop AI

The following is code to calculate the closed-loop system of Active Inference (AI) on the systems in Equation 2-2.

```
function sysclAI = closedLoopAI(sys,sysw,sysz,Pixtilde,Piytilde,Piutilde,
    Q,R,N)
%CLOSEDLOOPAI Closed-loop system of AI on a ss system
%
%   sysclAI = CLOSEDLOOPAI(sys,nw,nz,Pixtilde,Piytilde,Piutilde,p,Q,R,N)
%   gives the closed loop ss system sysclAI that results from running
%   Active Inference on the system sys, where the last nz inputs is the
%   measurement noise, the nw inputs before that the process
%   noise. sysw and sysz are the noise systems, having normalized WGN as
%   inputs and the noise w and z as output. Pixtilde and Piytilde are the
%   generalised precision matrices for the process and measurement (noise
    ),
%   respectively. Piutilde can be freely chosen, or set to Inf, in which
%   case mu_u is given by eta. Q, R and N are the normal LQR weighting
%   matrices.
```

```
%
%   Inputs:
%   sys:    System to be controlled, given by
%              .
%          x = Ax + Bu + Gw          {State equation}
%          y = Cx + Du + Hz          {Measurements}
%          such that sys = ss(A,[B G 0],C,[D 0 H])
%
%   sysw:     Process noise filter
%   sysz:     Measurement noise filter
%
%   Pixtilde:   Generalised precision matrix for the process
%   Piytilde:   Generalised precision matrix for the measurements
%   Piutilde:   Generalised precision matrix for inputs. Set to Inf for
    AI
%          with mu_u = eta
%
%   p:  Number of generalised states
%
%   Q:  Weighing matrix for states (x'Qx)
%   R:  Weighing matrix for inputs (u'Ru)
%   N:  Weighing matrix for states/inputs (x'Nu). Set to 0 if omitted.
%
%   Outputs:
%   sysclAI:  Tuned closed-loop of AI on system sys
%
%
%   Jesse Coehoorn
%   MSc Thesis Systems & Control and Mechanical Engineering

% System dimensions
n = size(sys.A,2);            % state dimension of system
qw = size(sysw.C,1);          % output dimension of process noise filter
qz = size(sysz.C,1);          % output dimension of measurement noise
    filter
m = size(sys.B,2)-qw-qz;      % controllable input dimension of system
q = size(sys.C,1);            % output dimension of system
p = min(length(Pixtilde)/n+1,length(Piytilde)/q);

% Make sure that precision matrices have the same number of generalised
% states
if p*n < length(Pixtilde)
    Pixtilde = Pixtilde(1:p*n,1:p*n);
else
    Pixtilde = blkdiag(Pixtilde,zeros(n*p-length(Pixtilde)));
end
if p*q < length(Piytilde)
    Piytilde = Piytilde(1:p*q,1:p*q);
else
    Piytilde = blkdiag(Piytilde,zeros(q*p-length(Piytilde)));
end

% Generalised matrices
```

```
Atilde = kron(eye(p),sys.A);        % Generalised A matrix
Btilde = kron(eye(p),sys.B(:,1:m)); % Generalised B matrix
Ctilde = kron(eye(p),sys.C);        % Generalised C matrix
Dtilde = kron(eye(p),sys.D(:,1:m)); % Generalised D matrix
Gtilde = [-sys.C/sys.A*sys.B(:,1:m)+sys.D(:,1:m);
          zeros(q*(p-1),m)];        % Generalised forward matrix
Dx = [zeros(n*p,n) ...
      [eye(n*(p-1)); zeros(n,n*(p-1))]]; % Derivative matrix for states

% Learning rates
kappax = realp('kappax',eye(n*p));  % State estimates learning rate
rho = realp('rho',eye(m));          % Control learning rate

if isinf(Piutilde) % the case where mu_u is set equal to eta

    % AI controller
    Aai = [...
    [Dx-kappax*Ctilde'*Piytilde*Ctilde-kappax*(Dx-Atilde)'*Pixtilde*(Dx-
        Atilde);
    rho*Gtilde'*Piytilde*Ctilde] ...
    zeros(n*p+m,m)];
    Bai = [kappax*Ctilde'*Piytilde ...
           -kappax*Ctilde'*Piytilde*Dtilde+kappax*(Dx-Atilde)'*Pixtilde*
               Btilde;
           -rho*Gtilde'*Piytilde rho*Gtilde'*Piytilde*Dtilde];
    Cai = [zeros(m,n*p) eye(m)];
    Dai = 0;
    sysAI = ss(Aai,Bai,Cai,Dai);


else % the case where we estimate the inputs

    % Obtain generalised matrices
    Du = [zeros(m*p,m) [eye(m*(p-1)); zeros(m,m*(p-1))]];

    % Extra tuning parameters
    kappau = realp('kappau',ones(m*p));      % Input estimates learning
        rate
    Piutilde = realp('Piutilde',Piutilde);  % Input precision matrix
    Piutilde.Free = Piutilde.Value;
    Piutilde.Minimum(Piutilde.Free) = eps;

    % AI controller
    Aai = [...
    [Dx-kappax*Ctilde'*Piytilde*Ctilde-kappax*(Dx-Atilde)'*Pixtilde*(Dx-
        Atilde) ...
    -kappax*Ctilde'*Piytilde*Dtilde+kappax*(Dx-Atilde)'*Pixtilde*Btilde;
    -kappau*Dtilde'*Piytilde*Ctilde+kappau*Btilde'*Pixtilde*(Dx-Atilde)
        ...
    Du-kappau*Dtilde'*Piytilde*Dtilde-kappau*Btilde'*Pixtilde*Btilde-
        kappau*Piutilde;
    rho*Gtilde'*Piytilde*Ctilde rho*Gtilde'*Piytilde*Dtilde] ...
    zeros((n+m)*p+m,m)];
    Bai = [kappax*Ctilde'*Piytilde zeros(n*p,m*p);
           kappau*Dtilde'*Piytilde kappau*Piutilde;
           -rho*Gtilde'*Piytilde zeros(m,m*p)];
    Cai = [zeros(m,(n+m)*p) eye(m)];
    Dai = 0;
    sysAI = ss(Aai,Bai,Cai,Dai);


end

% Output derivatives (approximated with high-pass filter)
s = tf('s');
d = [];
for gen_state = 0:p-1
    d = [d;(s/(1e-4*s+1))^gen_state*eye(q)];
end

% AI controller with derivatives
sysAId = sysAI*blkdiag(d,eye(m*p));

% Input and Output names. Append sys such that it also outputs the states
    ;
% used for tuning
sys = ss(sys.A,sys.B,[sys.C;eye(n)],[sys.D;zeros(n,m+qw+qz)]);
sys.y(1:q) = ionames('y',q);
sys.y(q+1:end) = ionames('x',n);
sys.u(1:m) = ionames('u',m);
sys.u(m+1:m+qw) = ionames('w',qw);
sys.u(m+qw+1:end) = ionames('z',qz);
sysAId.y = 'u';
sysAId.u(1:q) = ionames('y',q);
sysAId.u(q+1:end) = ionames('eta',p*m);

% Closed-loop system
sysw.u = 'omega';
sysw.y = 'w';
sysz.u = 'zeta';
sysz.y = 'z';
sysclAI = connect(sys,sysAId,sysw,sysz,...
                  [{'omega'};{'zeta'}],[{'y'};{'x'};{'u'}]);

% Tuning
try N; catch N = zeros(n,m); end
opt = systuneOptions('UseParallel',true,'RandomStart',19);
goal = TuningGoal.LQG([{'omega'};{'zeta'}],[{'x'};{'u'}],...
                      eye(qw+qz),[Q N;N' R]);
[sysclAI,~,~,~] = systune(sysclAI,goal,[],opt);

% Tuned closed-loop system
sysclAI = ss(sysclAI);


end
```

```


function ioname = ionames(ioname,n)

    if n == 1
        ioname = {ioname};
    else
        ioname = cellstr(ioname+"("+(1:n)'+")");
    end
end
```

A-3 Precision matrix

The following is code to calculate the precision matrix for different filters.

```
function Pi = filterPrecision(type,varargin)
%FILTERPRECISION Calculates precision matrix Pi for filtered white noise
%   Options:
%
%   Pi = FILTERPRECISION('tf',tf) calculates the precision matrix Pi for
%   transfer function tf
%
%   Pi = FILTERPRECISION('zp',zeros,poles,gain) calculates the precision
%   matrix Pi for the vector zeros, vector poles and optional scalar gain
%
%   Pi = FILTERPRECISION('coeff',den,num) calculates the precision matrix
%   Pi for the vector with denominator coefficients den and the vector
   with
%   numerator coefficients num
%
%   Pi = FILTERPRECISION('lp1',H(s),p) calculates the precision matrix Pi
%   for the onesided (causal) filter H(s) in Laplace domain, with p the
%   desired precision matrix size
%
%   Pi = FILTERPRECISION('lp2',H(s),p) calculates the precision matrix Pi
%   for the twosided (non-causal) filter H(s) in Laplace domain, with p
   the
%   desired precision matrix size
%
%   Pi = FILTERPRECISION('fou1',H(s),p) calculates the precision matrix
   Pi
%   for the onesided (causal) filter H(w) in Fourier domain, with p the
%   desired precision matrix size
%
%   Pi = FILTERPRECISION('fou2',H(s),p) calculates the precision matrix
   Pi
%   for the twosided (non-causal) filter H(w) in Fourier domain, with p
   the
%   desired precision matrix size
%
```

```matlab
%   Pi = FILTERPRECISION('time',h(t),p) calculates the precision matrix Pi
%   for the filter h(t) in time-domain, with p the desired precision matrix
%   size
%
%
%   Jesse Coehoorn
%   MSc Thesis Systems & Control and Mechanical Engineering


% setup symbolic variables
syms s          % Laplace variable
syms t real     % time


switch type
    case 'tf'

        % make symbolic transfer function
        H = varargin{1};
        num(s) = poly2sym(H.Numerator{1},s);
        den(s) = poly2sym(H.Denominator{1},s);
        Hs(s) = num/den;

        % obtain precision matrix size
        p = polynomialDegree(den)-polynomialDegree(num);

        % time-domain version of filter
        h(t) = simplify(ilaplace(Hs)*heaviside(t));

    case 'zp'

        % make symbolic transfer function
        z = varargin{1};
        p = varargin{2};
        try gain = varargin{3}; catch, gain = 1; end
        num(s) = prod(s-z);
        den(s) = prod(s-p);
        Hs(s) = gain*num/den;

        % obtain precision matrix size
        p = polynomialDegree(den)-polynomialDegree(num);

        % time-domain version of filter
        h(t) = simplify(ilaplace(Hs)*heaviside(t));

    case 'coeff'

        % make symbolic transfer function
        coeffp = varargin{1};
        coeffz = varargin{2};
        num(s) = poly2sym(coeffz,s);
        den(s) = poly2sym(coeffp,s);
        Hs(s) = num/den;

        % obtain precision matrix size
        p = polynomialDegree(den)-polynomialDegree(num);

        % time-domain version of filter
        h(t) = simplify(ilaplace(Hs)*heaviside(t));

    case 'lp1'

        % time-domain version of filter
        H = varargin{1};
        h(t) = simplify(ilaplace(H)*heaviside(t));

        % obtain precision matrix size
        p = varargin{2};

    case 'lp2'

        % time-domain version of filter
        H = varargin{1};
        h(t) = simplify(ilaplace(H));

        % obtain precision matrix size
        p = varargin{2};

    case 'fou1'

        % time-domain version of filter
        H = varargin{1};
        h(t) = simplify(ifourier(H,t)*heaviside(t));

        % obtain precision matrix size
        p = varargin{2};

    case 'fou2'

        % time-domain version of filter
        H = varargin{1};
        h(t) = simplify(ilaplace(H,t));

        % obtain precision matrix size
        p = varargin{2};

    case 'time'

        % time-domain version of filter
        h(t) = varargin{1};

        % obtain precision matrix size
        p = varargin{2};

```

```
end


% compute derivatives of the filter
dh = h(t);
for j=1:p-1
    dh = [dh; simplify(diff(dh(end)))];
end


% calculate variances and precision matrix
Sigma = zeros(p);
for j=1:p
    if mod(j,2) % row 1, 3, etc.
        for k=1:2:p % column 1, 3, etc.
            if k < j % mirror entries below the main diagonal
                Sigma(j,k) = Sigma(k,j);
            else
                Sigma(j,k) = vpaintegral(dh(k)*dh(j),-inf,inf);
            end
        end
    else % row 2, 4, etc.
        for k=2:2:p % column 2, 4, etc.
            if k < j % mirror entries below the main diagonal
                Sigma(j,k) = Sigma(k,j);
            else
                Sigma(j,k) = vpaintegral(dh(k)*dh(j),-inf,inf);
            end
        end
    end
end


% invert to obtain precision matrix
Pi = inv(real(Sigma));

end
```

A-4 Random SISO systems

The following is code to generate a random single-input, single-output (SISO) system within certain bounds.

```
function [sys,p,z] = rsiso(np,nz,prange,zrange,pcomplex,pdouble)
%RSISO Generate random SISO system
%
%   [sys,p,z] = RSISO(np,nz,prange,zrange,pcomplex,pdouble)
%
%   inputs:
%   np: number of poles
%   nz: number of zeros
```

```matlab
%   prange: [min max cmax], min and max value for real part of poles, max
%           value for complex part of poles
%   zrange: [min max cmax], min and max value for real part of zeros, max
%           value for complex part of zeros
%   pcomplex:   chance of complex pole/zero, default: 0.5
%   pdouble:    chance of doubling a pole/zero, default: 0.05
%
%   outputs:
%   sys: generated random system in ss form
%   p: poles of the system
%   z: zeros of the system
%
%
%   Jesse Coehoorn
%   MSc Thesis Systems & Control and Mechanical Engineering


try pcomplex; catch, pcomplex = 0.5; end % set default for pcomplex
try pdouble; catch, pdouble = 0.05; end % set default for pdouble


p = rpoles(np,prange,pcomplex,pdouble); % obtain poles
z = rpoles(nz,zrange,pcomplex,pdouble); % obtain zeros


sys = ss(zpk(z,p,1)); % set ss system


end


function p = rpoles(np,prange,pcomplex,pdouble)
% generate vector of poles (or zeros)


% functions for random numbers in the right ranges
prand = @() rand*(prange(2) - prange(1)) + prange(1);
pcrand = @() rand*prange(3);


% initialize
inp = 0;
p = [];


% fill the vector till the limit is reached
while inp < np
    if rand < pcomplex && inp < np-1 % complex pole/zero
        s = prand() + pcrand()*1i;
        p = [p s conj(s)];

        inp = inp + 2;

        while inp < np-1 % chance of doubling
            if rand < pdouble
                p = [p s conj(s)];

                inp = inp + 2;
            else
                break;
            end
        end

    else % real pole/zero
        p = [p prand()];

        inp = inp + 1;

        while inp < np % chance of doubling
            if rand < pdouble
                p = [p p(end)];

                inp = inp + 1;
            else
                break;
            end
        end
    end

end

end
```

A-5 Comparison

```matlab
%% Code for a realization of AI and extended LQG on a random system
%   The first block of code sets up the different systems. It generates a
%   random MIMO SS system using rss. After that, SISO systems for the
%   noises w and z are generated, one for each dimension. These are
%   appended to form a single noise system. Then, the precision matrices
%   are formed in the correct order.
%   The second block of code obtains the closed-loop of extended LQG on
%   these systems.
%   The third block of code obtains the closed-loop of AI on these systems.
%   The last block of code performs the simulation and computes the
%   incurred costs.
%
%   Jesse Coehoorn
%   MSc Thesis Systems & Control and Mechanical Engineering

clear variables; close all; clc;

```

```matlab

%% Setup systems

% controlled MIMO system
n = 1; % state dimension
m = 1; % input dimension
q = 1; % output dimension
sys = rss(n,q,m);
sys = ss(sys.A*randn(n),[sys.B eye(n) zeros(n,q)],...
         sys.C,[sys.D zeros(q,n) eye(q)]);
if any(abs(real(pole(sys))) < 1e-9)
    warning('System has pole close to 0, gives problems for AI')
end

% noise filters (poles -10 < p < 0, zeros -1 < z < 1)
% filter for process noise
nw = 6; % order of numerator
nzw = 0; % order of denominator
for i=1:n
    [sysw,p,z] = rsiso(nw,nzw,[-10 -1 10],[-1 1 1]);
    Piw = filterPrecision('zp',z,p);
    sysfw(i).sysw = sysw*sqrt(Piw(1,1));
    sysfw(i).Piw = Piw/Piw(1,1);
end
% filter for measurement noise
nz = 6; % order of numerator
nzz = 0; % order of denominator
for i=1:q
    [sysz,p,z] = rsiso(nz,nzz,[-10 -1 10],[-1 1 1]);
    Piz = filterPrecision('zp',z,p);
    sysfz(i).sysz = sysz*sqrt(Piz(1,1));
    sysfz(i).Piz = Piz/Piz(1,1);
end

% precision matrices
Piw = blkdiag(sysfw.Piw);
order = [];
for i=1:(nw-nzw)
    order = [order i:(nw-nzw):(n-1)*(nw-nzw)+i];
end
Piwtilde = zeros((nw-nzw)*n);
for i=1:(nw-nzw)*n
    for j=1:(nw-nzw)*n
        Piwtilde(i,j) = Piw(order(i),order(j));
    end
end
Piz = blkdiag(sysfz.Piz);
order = [];
for i=1:(nz-nzz)
    order = [order i:(nz-nzz):(q-1)*(nz-nzz)+i];
end
Piztilde = zeros((nz-nzz)*q);
for i=1:(nz-nzz)*q
```

```
    for j=1:(nz-nzz)*q
        Piztilde(i,j) = Piz(order(i),order(j));
    end
end

% noise filters
sysw = append(sysfw.sysw);
sysz = append(sysfz.sysz);


%% LQG system

% cost matrices
Q = eye(n);
R = eye(m);

% closed-loop LQG
[sysclLQG,~,p] = closedLoopLQG(sys,sysw,sysz,Q,R);

if max(real(pole(sysclLQG))) > 0
    warning('Closed-loop LQG system is unstable, probably due to the
       conversion to an explicit system not working properly (round-off
       errors, etc.)');
end


%% AI system

% amount of generalised states
p = min(p);

% initial Piutilde matrix
Piutilde = eye(m*p)*1e0;

% closed-loop AI
sysclAI = closedLoopAI(sys,sysw,sysz,Piwtilde,Piztilde,Piutilde,Q,R);

if max(real(pole(sysclAI))) > 0
    warning('Closed-loop AI system is unstable, probably due to failed
       tuning');
end


%% Simulation

% time vector
time = 0:0.001:1000;
nt = length(time);

% noise vector
w = randn(nt,n+q);

% costs incurred for LQG
```

```
Y = lsim(sysclLQG,w,time);
cost = zeros(2,nt);
for i=1:nt
    cost(1,i) = Y(i,q+1:q+n)*Q*Y(i,q+1:q+n)';
    cost(2,i) = Y(i,q+n+1:q+n+m)*R*Y(i,q+n+1:q+n+m)';
end
CLQG = sum(cost);

% costs incurred for AI
Y = lsim(sysclAI,w,time);
cost = zeros(2,nt);
for i=1:nt
    cost(1,i) = Y(i,q+1:q+n)*Q*Y(i,q+1:q+n)';
    cost(2,i) = Y(i,q+n+1:q+n+m)*R*Y(i,q+n+1:q+n+m)';
end
CAI = sum(cost);
```

Bibliography

[1] Brian D.O. Anderson and John B. Moore. *Optimal control: linear quadratic methods.* Prentice-Hall International, Inc., 1989.

[2] Ajith Anil Meera and Martijn Wisse. Free energy principle based state and input observer design for linear systems with colored noise. In *2020 American Control Conference (ACC)*, pages 5052–5058. IEEE, 2020.

[3] Mohamed Baioumy, Paul Duckworth, Bruno Lacerda, and Nick Hawes. Active inference for integrated state-estimation, control, and learning. *arXiv preprint arXiv:2005.05894*, 2020.

[4] Manuel Baltieri and Christopher L. Buckley. On kalman-bucy filters, linear quadratic control and active inference. *arXiv preprint arXiv:2005.06269*, 2020.

[5] Stephen Boyd, Laurent El Ghaoui, Eric Feron, and Venkataramanan Balakrishnan. *Linear matrix inequalities in system and control theory.* SIAM, 1994.

[6] Arthur E. Bryson and D.E. Johansen. Linear filtering for time-varying systems using measurements containing colored noise. *IEEE Transactions on Automatic Control*, 10(1):4–10, 1965.

[7] Jingjing Bu, Afshin Mesbahi, Maryam Fazel, and Mehran Mesbahi. LQR through the lens of first order methods: Discrete-time case. *arXiv preprint arXiv:1907.08921*, 2019.

[8] Christopher L. Buckley, Chang S. Kim, Simon McGregor, and Anil K. Seth. The free energy principle for action and perception: A mathematical review. *Journal of Mathematical Psychology*, 81:55–79, 2017.

[9] Xinyi Chen and Elad Hazan. Black-box control for linear dynamical systems. *arXiv preprint arXiv:2007.06650*, 2020.

[10] David R. Cox and Hilton D. Miller. *The theory of stochastic processes*, volume 134. CRC press, 1977.

[11] Ilyas Fatkhullin and Boris Polyak. Optimizing static linear feedback: Gradient method. *arXiv preprint arXiv:2004.09875*, 2020.

[12] Karl J. Friston. A theory of cortical responses. *Philosophical transactions of the Royal Society B: Biological sciences*, 360(1456):815–836, 2005.

[13] Karl J. Friston. Hierarchical models in the brain. *PLoS Comput Biol*, 4(11):e1000211, 2008.

[14] Karl J. Friston and Ping Ao. Free energy, value, and attractors. *Computational and mathematical methods in medicine*, 2012, 2012.

[15] Karl J. Friston, Jean Daunizeau, James Kilner, and Stefan J. Kiebel. Action and behavior: a free-energy formulation. *Biological Cybernetics*, 102(3):227–260, 2010.

[16] Karl J. Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of Physiology-Paris*, 100(1-3):70–87, 2006.

[17] Karl J. Friston, Nelson J. Trujillo-Barreto, and Jean Daunizeau. DEM: a variational treatment of dynamic systems. *Neuroimage*, 41(3):849–885, 2008.

[18] Sherin Grimbergen. The state space formulation of active inference: Towards brain-inspired robot control. Master's thesis, Delft University of Technology, 2019.

[19] Bruce Hajek. *Random processes for engineers*. Cambridge university press, 2015.

[20] Iris L. Hijne. Generalised motions in active inference by finite differences - active inference in robotics. Master's thesis, Delft University of Technology, 2020.

[21] Rudolf E. Kálmán. Contributions to the theory of optimal control. *Bol. Soc. Mat. Mexicana*, 5(2):102–119, 1960.

[22] Rudolf E. Kálmán. A new approach to linear filtering and prediction problems. *Trans. ASME Ser. D: J. Basic Eng.*, 82:35–45, 1960.

[23] Rudolf E. Kálmán and Richard S. Bucy. New results in linear filtering and prediction theory. *Trans. ASME Ser. D: J. Basic Eng.*, 83:95–108, 1961.

[24] Pablo Lanillos, Jordi Pages, and Gordon Cheng. Robot self/other distinction: active inference meets neural networks learning in a mirror, 2020.

[25] Guillermo Oliver, Pablo Lanillos, and Gordon Cheng. Active inference body perception and action for humanoid robots. *arXiv preprint arXiv:1906.03022*, 2019.

[26] Corrado Pezzato, Riccardo Ferrari, and Carlos H. Corbato. A novel adaptive controller for robot manipulators based on active inference. *IEEE Robotics and Automation Letters*, 5(2):2973–2980, 2020.

[27] Léo Pio-Lopez, Ange Nizard, Karl J. Friston, and Giovanni Pezzulo. Active inference and robot control: A case study. *Journal of the Royal Society Interface*, 13(122), 2016.

[28] Cansu Sancaktar, Marcel A.J. van Gerven, and Pablo Lanillos. End-to-end pixel-based deep active inference for body perception and action. In *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 1–8. IEEE, 2020.

[29] Max Simchowitz, Karan Singh, and Elad Hazan. Improper learning for non-stochastic control. *arXiv preprint arXiv:2001.09254*, 2020.

Glossary

List of Acronyms

3mE	Mechanical, Maritime and Materials Engineering
CoR	Cognitive Robotics
DCSC	Delft Center for Systems and Control
I	First comparison: Normal LQG and reduced AI White Gaussian Noise (WGN) setting with knowledge of inputs, performed in section 4-2. Relevant sections are subsection 2-2-1, section 2-3 and subsection 3-1-3. This leads to an equivalent expression for the two controllers.
II	Second comparison: Extended LQG and normal AI Linearly filtered WGN, without knowledge of inputs for AI, performed in section 4-3. Relevant sections are section 2-2, section 2-3 and subsection 3-1-1. An equivalent expression is no longer possible, but AI can mostly be tuned to obtain the same (optimal) LQG costs. This comparison is performed numerically, using MATLAB.
AI	Active Inference
ARE	Algebraic Riccatti Equation
DEM	Dynamic Expectation Maximization
FE	Free Energy
FEP	Free Energy Principle
LTI	Linear Time-Invariant
LTV	Linear Time-Varying
LQG	Linear-Quadratic Gaussian
LQR	Linear-Quadratic Regulator
MIMO	multiple-input, multiple-output
OO	Online Optimization
PSD	Power Spectral Density

SISO	single-input, single-output
WGN	White Gaussian Noise
WSS	Wide-Sense Stationary

List of Symbols

$\delta(t)$	Dirac delta function
κ_u	Input estimation learning rate
κ_x	State estimation learning rate
ω	Normalized WGN, input to Linear Time-Invariant (LTI) filter which forms w
Π	Block diagonal matrix with precisions on output noise, system noise and input priors
Π_w	Precision (inverse covariance) matrix of the process noise w
Π_z	Precision (inverse covariance) matrix of the measurement noise z
ρ	Control learning rate
$\rho(t)$	Normalized auto-correlation of a signal
$\tilde{\eta}$	Generalized prior belief on inputs
$\tilde{\mu} = \{\tilde{\mu}_x, \tilde{\mu}_u\}$	Generalized estimates, of states and inputs
$\tilde{\Pi}_u$	Precision matrix on the prior $\tilde{\eta}$
ζ	Normalized WGN, input to LTI filter which forms z
$\hat{A}$	(Desired) state matrix used in the forward model $\tilde{G}$
$\mathcal{D}$	Derivative operator for generalized variables, such that $\tilde{(\cdot)}' = \mathcal{D}\tilde{(\cdot)}$. $\mathcal{D}$ is thus a block matrix with identity matrices on the block super-diagonal and zeros otherwise
$\tilde{A} = I_{p+1} \otimes A$	Generalised state matrix
$\tilde{B} = I_{p+1} \otimes B$	Generalised input matrix
$\tilde{C} = I_{p+1} \otimes C$	Generalised output matrix
$\tilde{D} = I_{p+1} \otimes D$	Generalised feedthrough matrix
$\tilde{G}$	Generalised forward model
A	State matrix of a linear state-space system
B	Input matrix of a linear state-space system
C	Output matrix of a linear state-space system
D	Feedthrough matrix of a linear state-space system
F	Free Energy
I_n	Identity matrix of size n
K	Control gain
L	Estimator (Kalman) gain
m	System
P	Solution to the estimation Algebraic Riccatti Equation (ARE)

p	Number of derivatives for a generalised coordinate
$Q \geq 0$	State costs matrix
$R > 0$	Input costs matrix
S	Solution to the control ARE
$S(j\omega)$	The Power Spectral Density (PSD) of a signal
$u \in \mathbb{R}^m$	Inputs to a system
$w \in \mathbb{R}^n$	Process noise, zero-mean additive Gaussian distributed
$x \in \mathbb{R}^n$	States of a system
$y \in \mathbb{R}^q$	Outputs/observations/measurements of a system, available to the controller
$z \in \mathbb{R}^q$	Measurement noise, zero-mean additive Gaussian distributed
$\mathbb{E}[\cdot]$	Expectation of a signal
$\otimes$	Kronecker product
$f(\tilde{\mu})$	Our belief or desired belief of $\tilde{f}(\tilde{x}, \tilde{u})$
$g(\tilde{\mu})$	Our belief or desired belief of $\tilde{g}(\tilde{x}, \tilde{u})$