Hierarchical Active Inference for Continuous Robot Control

Goal-Directed Hierarchical Models for Active Inference Agents

by Victor van Vucht

Master of Science Thesis

For the degree of Master of science in Mechanical Engineering
at Delft University of Technology

To be defended on 29 January 2021 at 10:00

Supervisors:	Prof.dr.ir. Martijn Wisse Dr. André Prins
Thesis committee:	Prof.dr.ir. Martijn Wisse Dr. André Prins Dr. Arkady Zgonnikov Ir. Charel van Hoof

Cognitive Robotics

Acknowledgements

I would like to thank Martijn for his contagious fascination for this subject and the support he has provided throughout this challenging process. Our discussions always helped to highlight the essence of the issue. Thank you André for your incredible patience and time during the last stretch, and the perspective you offered on the topic and on the process. I would like to thank Goof for his dedicated supervision and feedback. Many thanks also to the Active Inference group for the different views on the research field, may you have many fruitful future meetings.
I am incredibly grateful to my parents for encouraging me throughout the ups and downs of this journey. Finally, thank you to Bas, Yanni and all my friends who have kept me company (but not too much company) during the pandemic.

21 January 2021 Victor van Vucht

Abstract

Active inference is a method for state estimation and control actions that is based on the Free Energy principle, which explains how biological agents infer the state of their environment and act upon it by maintaining a model of that environment and evaluating predictions. This method merges both action and sensory processing and is therefore a promising approach for robotic control. In active inference, the internal representations (or states) in the agent can be described by applying hierarchical dynamical models (HDMs). This offers a way to describe the dynamical development of states, and to structure them hierarchically. Despite the possibilities of this hierarchical approach, there are very few examples of hierarchical active inference being applied to robotics or control problems. This thesis aims to make a step in exploring the possibilities of active inference under a HDM. We describe the implementation of a control algorithm based on the hierarchical formulation of active inference on several continuous control problems, most importantly a 2-D robot arm. The algorithm is first demonstrated with a basic single level cart simulation, analysing the effect of the various parameters and inclusion of higher dynamic orders on the stability. A following simulation of multiple carts demonstrates a simple example of a hierarchical division of goals, and how high level objectives are realized by reaching lower level goals. It also shows how insolvable prediction errors at lower levels are propagated up the hierarchy. Finally, the simulation of a 2-D robot arm shows how these hierarchies can introduce goal-directed behaviour in practical control problems. We made several attempts to design a hierarchical generative model capable of realizing position-reaching behaviour in the robot arm, dictating joint angles from desired positions, with varying levels of success. Several pitfalls were encountered in choosing a suitable model for the control task. Most importantly, it was found that the convergence was faulty when higher levels contained more independent states. This emphasizes the information-reducing role of hierarchies. Lastly, we demonstrate that the hierarchical generative model is capable of adding complexity to the robots behaviour, by expanding the goal-reaching objective with a path-tracing task. In short, hierarchical active inference can be applied effectively to the demonstrated goal-reaching control problems, however this requires a careful consideration of the generative model for the task at hand.

Contents

Chapter 1

Introduction

Robots are effectively deployed for many tasks. Their common merits are that they are precise, strong and can do the same thing over and over again for a long time. This is useful because for us humans, long repetitive work usually results in boredom and physical stress. At the same time, the task-consistency of robots is a disadvantage. In many cases they are only effective when the conditions of their task remain the same. The strength of natural organisms such as animals, and in particular humans, is that they can quickly adapt to changing environments. Specifically, they can gather information about the world around them and adapt their behaviour accordingly.

Science has made a lot of progress in bridging the gap between machine behaviour and natural cognitive functions. Developments in the field of machine learning, neural networks and artificial intelligence in general have shown the powerful abilities of machines to process information and recognize patterns. However, they require very large amounts of information and are typically restricted to a single task [7].

Robots today are still a long way away from being as physically and behaviourally adaptable as humans. That is why one of the big challenges for the scientific community is to develop more adaptive behaviour in robotics to handle different situations based on gathered information. This will enable robots to gain a more natural intelligence that will help them interact with a dynamic world.

Principles that explain intelligence and behaviour in natural organisms provide useful tools that help to realize these same characteristics in machines. Active inference is one such concept. Proposed by neuroscientist Karl Friston, active inference provides a theoretical basis for interactions between natural beings and their environments. It describes how biological agents (humans, animals and other organisms) keep an equilibrium with their environment by processing sensory information and performing actions on the external world. In order to do this, agents maintain an internal model of their environment by minimizing the so-called Free Energy, which is a measure of uncertainty about external states. The actions that are performed based on this model helps the agent to gain control over its situation. This is an interesting concept for robotics, as robots are also independent agents that must interact with their surroundings, and must exert some form of control over their environment.

One of the intriguing observations that have been made in literature on active inference is that the models can have a hierarchical form. This is important because many patterns and phenomena in nature can also be explained using hierarchy. Most recognisable objects and events are composed of smaller elements. For example, a forest is a collection of trees, which consist of many branches that in turn bear many leaves. This structure is also common in the temporal domain. In music for example, long sequences of simple notes and sounds produce the experience of listening to a song. In fact, sound itself is a complex series of acoustic vibrations that are perceived in terms of pitch and timbre. As organisms, we do not store and consciously process all the information that

is presented to us. Instead, we are able to summarize the essence of the vast volume of sensory signals into manageable pieces.

Hierarchy is also a present in the actions we take. When communicating with others, we consciously consider the ideas that we are trying to convey, however we are seldom aware of how we construct sentences, much less the complex actions our vocal system must take to produce the actual sounds. The key idea here is that cognitive function involves multiple levels of information processing that become increasingly abstract. There is ample evidence that human and animal brains facilitate this hierarchy for processing information [8] [19]. The fact that active inference also supports hierarchical processing makes it an excellent candidate for developing efficient behaviour in robotics and robot control.

Hierarchical dynamical models in conjunction with the free energy principle have been thoroughly described in literature, most notably by Karl Friston [10] . Despite this, there seems to be little evidence of hierarchical models in active inference being explored as a control method. There is some work that demonstrates the merits of hierarchical models for perception only [17] [24], and some applications in robotics that incorporate sensory integration using active inference [21]. Also, hierarchy and active inference have been applied to some control cases in a machine learning context [20]. However, a dedicated investigation of hierarchical models in active inference for simple, continuous control applications seems to be missing, which is where this thesis contributes.

This report will be analysing the implementation of an active inference based control algorithm using hierarchical models. The motivation for this investigation is exploring the possibilities of this approach for robotics, therefore it is appropriate to have a test case that reflects a practical robot control application. The subject of the control algorithm for this thesis is chosen to be a two-dimensional robot arm. This is considered to be simple enough to demonstrate the basic functionality of the algorithm but includes certain important characteristics such as non-linearity that are relevant for real-world applications.

In brief, the goal of this thesis is to provide a hierarchical implementation of active inference on a 2D robot arm. This goal will be achieved by tackling the following sub-goals:

1. Present a concise description of the theoretical background of active inference.
2. Provide an explanation of how a hierarchical model can function within active inference.
3. Put a hierarchical active-inference based control algorithm into executable code for simulations, that can implement various internal and external models.
4. Demonstrate the algorithm for a single-level linear goal-reaching control problem.
5. Demonstrate the algorithm for a linear goal reaching control problem using a hierarchical generative model.
6. Apply the algorithm to a 2-D robot arm with various hierarchical generative models to investigate the function of the hierarchical model for active inference in robot control.

Chapter 2

Active inference & Hierarchy

Before we can tackle the concept of hierarchy in active inference, there must first be a basic understanding of the foundations of active inference as a principle. First the general concept of prediction error minimization as a derivation of the Free Energy principle will be handled, then expanded to a hierarchical formulation in 2.3. How this leads to action and perception is explained in section 2.4 thereby the first two of our goals are tackled:

Subgoal 1: Present a concise description of the theoretical background of active inference.

Subgoal 2: Provide an explanation of how a hierarchical model can function within active inference.

The theoretical basis for active inference is the Free Energy Principle. This chapter will start with an explanation of the Free energy principle, and how active inference emerges from this. The complete theory and related topics are very extensive, but here a summary will be given of the most important concepts for the purpose of our investigation.

The Free Energy Principle was proposed by Karl Friston to explain how natural organisms handle the perception of the outside world and react to changes in their environment [12]. By minimizing a measure called the Free Energy, they can be more certain about their situation and about their survival. Importantly, the organism also has the possibility to adapt the outside world to reduce this free energy, and this leads to the theory of Active Inference.

2.1 Free energy theory

2.1.1 How Organisms Minimize Entropy

The Free Energy principle starts with the idea that natural organisms need to occupy certain states in the world, and not others, in order to survive. A humans core temperature needs to be between 36 °C and 37 °C, for example. And a fish needs to be in water to survive. All organisms need nourishment, and need to place themselves in situations where they will obtain it. The probability of being in certain states and not others can be measured in informational entropy. If the entropy is low, there is a high probability of being in certain states. If the entropy is high, the probability is more "spread out" and we can not be so certain of the situation.

The differential entropy is a measure of the certainty of the states.

$$H = -\int_0^\infty p(x) ln(p(x))\,\mathrm{d}x \tag{2.1}$$

The differential entropy is the expected value of the surprise $-ln(p(x))$. Under ergodic assumptions, this is the same as the time-average of the surprise [14]. An ergodic stochastic process is one in

which the probability distribution over time is the same as the probability distribution at any given moment. This means that no choices are made that exclude certain states or change the probability distribution permanently. This is quite a drastic assumption, and might exclude forms of decision making that take the future into account.

$$I = -ln(p(x)) \tag{2.2}$$

The idea is then to maximize the probability of being in a certain state $p(x)$ at any given time and thereby minimizing surprisal. The problem for most systems, however, is that the state x is not readily available. Only certain sensory states s can be observed that are in some way related to the real-world states x. Our system must now find the conditional density $p(x|s)$. The probability of a certain state x_i existing when a certain s_j is observed is then $p(x_i|s_j)$ Bayes rule shows us how that this can be rewritten.

$$p(x_i|s_j) = \frac{p(s_j|x_i)p(x_i)}{p(s_j)} = \frac{p(s_j|x_i)p(x_i)}{\int_{i=0}^{\infty} p(s_j|x_i)p(x_i)\,\mathrm{d}x} \tag{2.3}$$

This still poses a problem, as this requires calculating the probability of observing a certain sensory state ($p(s_j)$) for every instance of the real-world states x. This is the integral that can be seen in the denominator in 2.3. This integral is most often intractable, meaning it would take too much effort to compute because there are many states x_i and exponentially many combinations of x.

Instead, we can use a method called variational Bayesian inference [4]. This method assumes a set of candidate distributions $q(x)$ can be used to approximate the exact conditional $p(x|s)$. These candidate distributions are usually of a simple form, and the issue is finding the candidate distribution $q(x)$ that best approximates the actual distribution $p(x|s)$. The difference between these two distributions can be gauged by a measure called the Kullbeck-Leibler divergence.

$$D_{KL}(q(x)||p(x|s)) = \int q(x)ln(\frac{q(x)}{p(x|s)}\,\mathrm{d}x \tag{2.4}$$

The Kullbeck-Leibler divergence describes how (un)alike two distributions are. When the KL-Divergence is zero, the two distributions are equivalent. The KL divergence can be split into another form.

$$\begin{aligned} D_{KL}(q(x)||p(x|s)) &= \int q(x)ln\left(\frac{q(x)}{p(x|s)}\right)\mathrm{d}x + ln(p(s)) \\ \mathcal{F} &= \int q(x)ln\left(\frac{q(x)}{p(x,s)}\right)\mathrm{d}x \\ \mathcal{F} &= -ln(p(s)) + D_{KL}(q(x)||p(x|s)) \end{aligned} \tag{2.5}$$

Notice that after this rearrangement, one of the terms is the surprise $-ln(p(s))$. This shows that what we call the *Free Energy* $\mathcal{F}$ [1] is an upper bound on the surprise. The difference between the surprise and the $\mathcal{F}$ is the Kullback-Leibler divergence of the approximate density and the real density.

[1] In some literature the Free Energy is the negative of the expression given here and forms a *lower* bound on the surprise. In those instances the objective is to maximize the free energy, whereas here we aim to minimize it.

This effectively means that although we want to minimize the surprise $-ln(p(s))$ we cannot do so directly because of a discrepancy between the real probability distribution of the states and the one we are using to approximate it ($D_{KL}(q(x)||p(x|s))$). But if we can minimize the Free Energy $\mathcal{F}$ the surprise will always be smaller.

A concept that is often encountered in literature on the free energy principle is the mean-field approximation. It is mentioned here for the interested reader but is not considered in this thesis. The mean-field approximation is meant to finesse optimization and it assumes that the states x can be subdivided into independent sets $x = [u, \theta, \gamma]$. The states u change relatively quickly in time and the parameters $[\theta, \gamma]$ describe more static properties of the distribution. The distribution $q(x)$ then becomes a product of the independent distributions $p(x) = p(u)p(\theta)p(\gamma)$. DEM,(Dynamic Expectation Maximization) [9] is a very general form of Free Energy minimization which considers these different sets an includes a process of learning on the parameters. Often, as in this thesis, uncertainty about the parameters $[\theta, \gamma]$ is ignored and only inference on the dynamic states is considered. Therefore we will only be considering a single set of states x for the distribution $q(x)$.

The objective is to minimize the free energy $\mathcal{F}$. To make this feasible, the Free Energy in 2.5 is further simplified using the Laplace Approximation. The laplace approximation involves expressing $q(x)$ as a gaussian distribution, parametrized by the mean and the variance (μ, σ).

$$q(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma}} \tag{2.6}$$

It is assumed that $q(x)$ is sharply peaked at its mean μ and that that the function $-ln(p(x, s))$ is smooth. The integrals in $\mathcal{F}$ can then be simplified by using a Taylor expansion of $-ln(p(x, s))$ around the point $x = \mu$. How the Laplace approximation is applied to the free energy and what the consequences of this simplification is can be quite difficult to grasp. It is discussed in the context of variational Free Energy in [13]. Perhaps a more extensive discussion of these mathematical methods is given in [6]. By using the Laplace approximation, a simplified approximation for the Free Energy is given that depends only on the means μ and the sensory signals s 2.7.

$$\mathcal{F} \approx -ln(p(s, \mu)) \tag{2.7}$$

2.2 Generative Model & Prediction Error

The density in used in the approximate Free Energy 2.7 can be split into parts:

$$\mathcal{F} = -ln(p(s|\mu)p(\mu)) \tag{2.8}$$

$p(s|\mu)$ describes the density of the sensory states based on some cause μ, and $p(\mu)$ describes the belief of the agent about the states that cause these sensory observations. These two distributions are assumed to be gaussian and their means and variance are described by the internal model of the agent.

These probability distributions are based on an assumption made by the agent that the real-world states develop based on a *generative model*. When considering continuous systems, the generative model can be described as a state space system (2.9) with a differential equation and noise terms.

$$\begin{aligned} \dot{x} &= f(x) + \omega \\ s &= g(x) + z \end{aligned} \tag{2.9}$$

Here, $g(x)$ is a static mapping that produces the sensory observations s from the hidden states x. The function $f(x)$ describes the dynamical development of the states x by describing its derivative in time $\dot{x}$. ω and z are noise terms, that describe random fluctuations on the states that add disturbances to f and g respectively.

Formulations of active inference in continuous time employ a concept called *generalized coordinates* [2]. This term should not be confused with generalized coordinates as used in analytical mechanics, where it provides a unique description of the configuration of a system. In active inference, the generalized coordinates of a certain state are all the time derivatives of that state (up to a certain practical point). Here, $\tilde{}$ denotes generalized coordinates. This means that $\tilde{x}$ is a vector containing x and its derivatives, $[x, \dot{x}, \ddot{x}, \dddot{x} ....]$ up to a certain "embedding order" meaning the highest practical derivative that is considered.

$$\begin{aligned} \dot{\tilde{x}} &= f(\tilde{x}) + \tilde{\omega} \\ \tilde{s} &= g(\tilde{x}) + \tilde{z} \end{aligned} \tag{2.10}$$

$$\begin{aligned} D\tilde{x} &= f(\tilde{x}) + \tilde{\omega} \\ \tilde{s} &= g(\tilde{x}) + \tilde{z} \end{aligned} \tag{2.11}$$

Equation 2.10 shows the state space model in generalized coordinates. Equation 2.11 describes the same system but this time uses the derivative operator D. D is a matrix that shifts the elements in $\tilde{x}$ in such a way that $\dot{\tilde{x}} = D\tilde{x}$. Notice that here we also take the derivatives of the noise terms. This means we assume that the noise is continuous and differentiable. Another way to describe this is that noise is "smooth".

It is assumed that the noise terms $\tilde{\omega}$ and $\tilde{z}$ have a Gaussian distribution. When the noise terms are Gaussian, a distribution of the generalized sensory signals and states $p(s, x)$ can be determined.

$$\begin{aligned} p(\tilde{s}, \tilde{x}) &= p(\tilde{s}|\tilde{x})p(\tilde{x}) \\ p(\tilde{s}|\tilde{x}) &= \mathcal{N}(\tilde{s} : g, \tilde{\Sigma}_z) \\ p(\tilde{s}|\tilde{x}) &= \mathcal{N}(D\tilde{x} : f, \tilde{\Sigma}_\omega) \end{aligned} \tag{2.12}$$

Equation 2.12 shows the normal (Gaussian) distribution obtained with a model using the functions f and g. The covariances $\tilde{\Sigma}_z$ and $\tilde{\Sigma}_\omega$ describe the influence of the signal noise $\tilde{z}$ and the dynamic noise $\tilde{\omega}$ respectively. The natural logarithm of this distribution then becomes [10]:

$$ln(p(\tilde{s}, \tilde{x})) = \frac{1}{2}(\tilde{s} - g)^T \Pi_z (\tilde{s} - g) + \frac{1}{2}(D\tilde{x} - f)^T \Pi_\omega (D\tilde{x} - f) - \frac{1}{2} ln(\Pi_z \Pi_\omega) \tag{2.13}$$

The precisions Π_z, Π_ω are the inverse of their respective covariance matrices ($\Pi = \Sigma^{-1}$). The density $ln(p(\tilde{s}, \tilde{x}))$ is the same one seen in the expression of the Free Energy 2.5. Remember that after having applied the Laplace Approximation, the Free energy is simplified to this density, where the states x are quantified by their means μ. From 2.13, the approximated Free Energy becomes:

$$\mathcal{F} = \frac{1}{2}\varepsilon_z^T \Pi_z \varepsilon_z + \frac{1}{2}\varepsilon_\omega^T \Pi_\omega \varepsilon_\omega + \frac{1}{2} ln(|\Pi_z| \cdot |\Pi_\omega|) \tag{2.14}$$

$$\begin{aligned} \varepsilon_z &= \tilde{s} - g(\tilde{\mu}) \\ \varepsilon_\omega &= D\tilde{\mu} - f(\tilde{\mu}) \end{aligned} \tag{2.15}$$

The Free energy now has the same form as 2.13 but the states are represented by their expected means $\tilde{\mu}$. These means represent the beliefs that the agent has about the states x. ε_z and ε_ω are the prediction errors on the sensory states $\tilde{s}$ and the beliefs $\tilde{\mu}$ respectively. They quantify how well the sensory states and the agents beliefs correspond to the predictions made by the model through the functions f and g As we can see, the Free Energy has become the sum of the squared prediction errors, weighted by the precisions Π.

2.3 Hierarchical World & Model

The previous section has discussed the Form of the Free Energy assuming a dynamic generative model. An important concept in this thesis is that this dynamic model can be cast in a hierarchical form, which allows the agent to structure it's beliefs in a hierarchical manner.

$$\begin{aligned} \dot{\tilde{x}} &= f(\tilde{x}) + \tilde{\omega} \\ \tilde{s} &= g(\tilde{x}) + \tilde{z} \end{aligned} \tag{2.16}$$

$$\begin{aligned} \dot{\tilde{x}}_m &= f_m(\tilde{x}_m, \tilde{v}_m) + \tilde{\omega}_m \\ \tilde{v}_{m-1} &= g_m(\tilde{x}_m, \tilde{v}_m) + \tilde{z}_m \\ &\vdots \\ \dot{\tilde{x}}_2 &= f_2(\tilde{x}_2, \tilde{v}_2) + \tilde{\omega}_2 \\ \tilde{v}_1 &= g_2(\tilde{x}_2, \tilde{v}_2) + \tilde{z}_2 \\ \dot{\tilde{x}}_1 &= f_1(\tilde{x}_1, \tilde{v}_1) + \tilde{\omega}_1 \\ \tilde{s} &= g_1(\tilde{x}_1, \tilde{v}_1) + \tilde{z}_1 \end{aligned} \tag{2.17}$$

2.16 shows the dynamic model that was considered in the previous section. This is expanded hierarchically as in 2.17 by introducing the causal states $\tilde{v}$. These causal states are used by the functions f_1 and g_1, but they are produced by another function from a higher level g_2. In turn, the second hierarchical level includes causal states $\tilde{v}_2$ that are determined by a level above that. This continues for however many levels m are present. This enables the separation of the model into different hierarchical levels that affect the lower levels through the causal states $\tilde{v}$.

As was discussed before in the previous section, the noise terms are assumed to behave according to a Gaussian distribution. This means that this generative model can be used to describe the

density $p(\tilde{s}, \tilde{x}, \tilde{v})$ as a product of a set of Gaussian distribution. Similar to 2.12, we can write:

$$\begin{aligned} p(\tilde{s}, \tilde{x}, \tilde{v}) &= p(\tilde{s}|\tilde{x}_1, \tilde{v}_1) \cdot p(\tilde{x}_1|\tilde{v}_1) \cdot p(\tilde{v}_1|\tilde{x}_2, \tilde{v}_2) \cdot p(\tilde{x}_2|\tilde{v}_2) \cdot p(\tilde{v}_2|\tilde{x}_3, \tilde{v}_3) \ldots \cdot p(\tilde{v}_m) \\ p(\tilde{x}_i|\tilde{v}_i) &= \mathcal{N}(D\tilde{x}_i : f(\tilde{x}_i, \tilde{v}_i), \tilde{\Sigma}_\omega) \\ p(\tilde{v}_i|\tilde{x}_{[i+1]}, \tilde{v}_{[i+1]}) &= \mathcal{N}(s : g(\tilde{x}_{[i+1]}, \tilde{v}_{[i+1]}), \tilde{\Sigma}_z) \\ p(\tilde{s}|\tilde{x}_1, \tilde{v}_1) &= \mathcal{N}(s : g(\tilde{x}_1, \tilde{v}_1), \tilde{\Sigma}_z) \end{aligned} \tag{2.18}$$

This derivation can be found in [10]. It is apparent from 2.18 that the probability distribution of the sensory observations $\tilde{s}$ is calculated by chaining together the probability distributions of the states $\tilde{x}$ and the causal states $\tilde{v}$. Again the derivative operator D shifts the vector of generalized coordinates $\tilde{x}$ in such a way that the derivative $\dot{\tilde{x}}$ is obtained. The resulting density consists of different distributions defined in each hierarchical level i up to a certain level m, as seen in equations 2.18 and 2.19.

$$p(\tilde{s}, \tilde{x}, \tilde{v}) = p(\tilde{s}|\tilde{x_1}, \tilde{v}_1) \cdot \prod_{i=1}^{m} (p(\tilde{x_i}|\tilde{v}_i) \cdot p(\tilde{v}_i|\tilde{x}_{i+1}, \tilde{v}_{i+1})) \tag{2.19}$$

For the Free Energy this distribution is expressed in terms of the means $[\tilde{\mu}, \tilde{v}]$ of the states $[\tilde{x}, \tilde{v}]$[2]and the natural logarithm is taken. As was previously seen for the single level dynamical case, the Free Energy F becomes a sum of prediction errors $(\varepsilon_\omega, \varepsilon_z)$. However, this time the free energy is based on a hierarchical model. The dynamical prediction errors ε_ω are the differences between the generative functions f and the corresponding beliefs about the dynamic states $\tilde{\mu}$. The causal prediction error are the differences between the functions g and the beliefs about the causal states $\tilde{v}$.

The Free Energy can be expressed in matrix form as in equation 2.20. The matrix form consists of the prediction errors in vector form and the precision matrices, which are the inverse covariance matrices ($\Pi = \Sigma^{-1}$).

$$F = -ln(p(\tilde{s}, \tilde{\mu}, \tilde{v})) = \frac{1}{2}\varepsilon_\omega^T \cdot \Pi_\omega \cdot \varepsilon_\omega + \frac{1}{2}\varepsilon_z^T \cdot \Pi_z \cdot \varepsilon_z + \frac{1}{2}ln(|\Pi_\omega \cdot \Pi_z|) \tag{2.20}$$

The matrix form of the free energy can be further compacted by merging the precision matrices together and stacking the prediction error vectors. This results in an equation consisting of just one vector ε and one matrix Π. The content of these two components is shown in more detail in 2.22 and 2.23. Note that the elements of ε are themselves vectors of generalized coordinates consisting of multiple dynamic orders[3], as denoted by $\tilde{}$. Also note that the elements in the precision matrix are themselves matrices which can have non-diagonal elements. This is because they encode

[2]It would be more correct to express the means of the dynamic and causal states $\tilde{x}$ and $\tilde{v}$ as $\tilde{\mu}_x$ and $\tilde{\mu}_v$ respectively. This is most often seen in the literature. However, here the choice has been made to use $\tilde{\mu}$ and $\tilde{v}$ instead, to make a clearer distinction between the two types of states.

the behaviour of (smooth) noise, which can include correlations between different dynamical orders.

$$F = -ln(p(\tilde{s}, \tilde{\mu}, v)) = \frac{1}{2}\varepsilon^T \cdot \Pi \cdot \varepsilon + \frac{1}{2} ln(|\Pi|) \tag{2.21}$$

$$\varepsilon = \begin{bmatrix} \varepsilon_v \\ \varepsilon_{\tilde{\mu}} \end{bmatrix} = \begin{bmatrix} \varepsilon_{z_1} \\ \varepsilon_{z_2} \\ \varepsilon_{z_3} \\ \vdots \\ \varepsilon_{z_m} \\ \varepsilon_{\omega_1} \\ \varepsilon_{\omega_2} \\ \vdots \\ \varepsilon_{\omega_m} \end{bmatrix} = \begin{bmatrix} \tilde{s} \\ v_1 \\ v_2 \\ \vdots \\ v_{m-1} \\ D\tilde{\mu}_1 \\ D\tilde{\mu}_2 \\ \vdots \\ D\tilde{\mu}_m \end{bmatrix} - \begin{bmatrix} g_1(\tilde{\mu}_1, v_1) \\ g_2(\tilde{\mu}_2, v_2) \\ g_3(\tilde{\mu}_2, v_2) \\ \vdots \\ g_m(\tilde{\mu}_m, v_m) \\ f_1(\tilde{\mu}_1, v_1) \\ f_2(\tilde{\mu}_2, v_2) \\ \vdots \\ f_m(\tilde{\mu}_m, v_m) \end{bmatrix} \tag{2.22}$$

$$\Pi = \begin{bmatrix} \Pi_z & \\ & \Pi_\omega \end{bmatrix} = \begin{bmatrix} \Pi_{z1} & & & & & & & & & \\ & \Pi_{z2} & & & & & & & & \\ & & \Pi_{z3} & & & & & & & \\ & & & \ddots & & & & & & \\ & & & & \Pi_{zm} & & & & & \\ & & & & & \Pi_{\omega 1} & & & & \\ & & & & & & \Pi_{\omega 2} & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & \Pi_{\omega m} \end{bmatrix} \tag{2.23}$$

2.4 Action & Perception

Now that an understandable expression for the approximation of the Free Energy has been obtained we can now turn to the matter of minimizing this quantity. It will become apparent that there are two ways in which the agent can minimize its Free Energy, namely by adapting its internal states that predict the sensory information, or by adjusting the sensory information itself through actions.

2.4.1 Prediction Error Minimization

As we have seen in 2.14, after having made many assumptions, the Free Energy can be described in terms of precision-mediated prediction errors. The goal of the agent is to minimize the Free Energy. We then have an optimization problem, to obtain the right internal beliefs $\tilde{\mu}$ and sensory

[3]Note that here we have let go of the generalized coordinates for the causal states v. Throughout this thesis we have assumed that since the causal states are not described by a dynamic function, as the states x are, they do not have to be described as multiple derivatives. However, this was an oversight. Since the states v are not static it is still relevant to express them in generalized form. This is not of great significance for the simulations throughout this thesis, as the generalized coordinates have a very small effect for the cases that are considered.

states $\tilde{s}$ to minimize the free energy $\mathcal{F}$:

$$(\tilde{\mu}, \tilde{s}) = \arg\min_{\mu,s} \mathcal{F} = \arg\min_{\mu,s} \frac{1}{2}\varepsilon_z^T \Pi_z \varepsilon_z + \frac{1}{2}\varepsilon_\omega^T \Pi_\omega \varepsilon_\omega + \frac{1}{2} ln(|\Pi_\omega \cdot \Pi_z|) \tag{2.24}$$

$$\varepsilon_z = \tilde{s} - g(\tilde{\mu}) \tag{2.25}$$

$$\varepsilon_\omega = D\tilde{\mu} - f(\tilde{\mu}) \tag{2.26}$$

$$\Pi_z = \begin{bmatrix} \sigma_{z[1,1]} & \sigma_{z[1,2]} & \cdots & \\ \sigma_{z[2,1]} & \sigma_{z[2,2]} & & \\ \vdots & & \sigma_{z[3,3]} & \\ & & & \ddots \end{bmatrix}^{-1} = \begin{bmatrix} \pi_{z[1,1]} & \pi_{z[1,2]} & \cdots & \\ \pi_{z[2,1]} & \pi_{z[2,2]} & & \\ \vdots & & \pi_{z[3,3]} & \\ & & & \ddots \end{bmatrix} \tag{2.27}$$

$$\Pi_\omega = \begin{bmatrix} \sigma_{\omega[1,1]} & \sigma_{\omega[1,2]} & \cdots & \\ \sigma_{\omega[1,2]} & \sigma_{\omega[2,2]} & & \\ \vdots & & \sigma_{\omega[3,3]} & \\ & & & \ddots \end{bmatrix}^{-1} = \begin{bmatrix} \pi_{\omega[1,1]} & \pi_{\omega[1,2]} & \cdots & \\ \pi_{\omega[1,2]} & \pi_{\omega[2,2]} & & \\ \vdots & & \pi_{\omega[3,3]} & \\ & & & \ddots \end{bmatrix} \tag{2.28}$$

$$\tag{2.29}$$

The agent can minimize the Free Energy by updating its internal belief $\tilde{\mu}$. Another way it can minimize the Free Energy is by performing an action on the world that will evoke different sensory observations $\tilde{s}$. The agent can therefore adjust $\tilde{\mu}$ and $\tilde{s}$. There are many ways that the agent can try to find $\tilde{\mu}$ and $\tilde{s}$ that minimize $\mathcal{F}$. However, one of the simplest ways is to perform *gradient descent*. Gradient descent methods calculate the gradient of a function to be minimized for certain arguments. This gradient describes the way that the function changes locally with the arguments, and so it dictates the direction the arguments should be changed in order to decrease the value of the function. A general gradient descent algorithm looks like this:

$$\dot{x} = -\kappa \nabla f(x) \tag{2.30}$$

κ is the learning rate, $f(x)$ is the function to be minimized, ∇ is an operator that denotes the Gradient of $f(x)$ and x is the argument to be adjusted. We can apply the same gradient descent scheme for the free energy. The appropriate update for the internal belief $\tilde{\mu}$ is found by:

$$\dot{\tilde{\mu}} = D\tilde{\mu} - \kappa_\mu \cdot \nabla_{\tilde{\mu}} \mathcal{F}(\tilde{\mu}, \tilde{s}) \tag{2.31}$$

$$= D\tilde{\mu} - \kappa_\mu \cdot \left[\varepsilon_z \Pi_z \frac{\delta \varepsilon_z}{\delta \tilde{\mu}} + \varepsilon_\omega \Pi_\omega \frac{\delta \varepsilon_\omega}{\delta \tilde{\mu}} \right] \tag{2.32}$$

This update of internal beliefs is a form of inference. With the help of gradient descent, the agent can find a local minimum of the free energy. The internal beliefs $\tilde{\mu}$ that correspond with this minimum are then the best guess that the agent can make of the causes of its sensory observations. Important to note here is that these internal beliefs will approach the real-world states, but only if the local minimum is also a global minimum, and if the internal model of the agent perfectly represents the outside world. However, the internal model may differ from the way the outside world is structured.

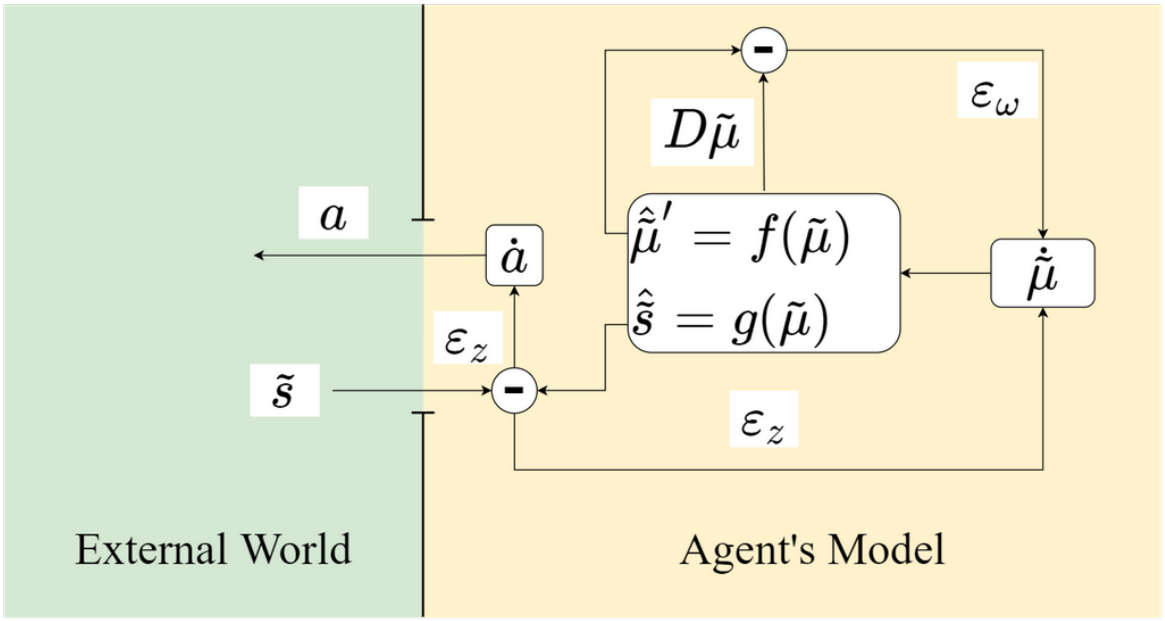


Figure 2.1: The only information that the agent has about the external environment is obtained through the sensory information $\tilde{s}$, and it can only influence the world through the actions a. The agent must assume the world behaves according to a model that is described by the functions f and g. These functions generate predictions about the dynamics of the world and the resulting observations based on a set of internal beliefs $\tilde{\mu}$. These beliefs are constantly updated based on the error of these predictions to better reflect the assumed dynamics and the sensory observations.

2.4.2 Action

As was said before, the agent can also minimize the Free Energy by performing actions on the world that result in sensory observations that are predicted by the internal model. This puts the "Active" in active inference. By doing this, the agent minimizes the prediction error $\varepsilon_z = \tilde{s} - g(\tilde{\mu})$. The appropriate change in actions can also be found with the gradient descent scheme.

$$\dot{a} = -\kappa_a \cdot \frac{\delta s}{\delta a} \nabla_s \mathcal{F}(\tilde{\mu}, \tilde{s}) \tag{2.33}$$

$$= -\kappa_a \cdot \frac{\delta s}{\delta a} \left[\varepsilon_z^T \Pi_z \frac{\delta \varepsilon_z}{\delta s} \right] \tag{2.34}$$

This requires a mapping $\frac{\delta s}{\delta a}$ that relates the actions to the sensory states. This is essentially an inverse model that determines the change that is needed in the actions a to achieve a change in the sensory observations $\tilde{s}$. In this way, actions are generated from the prediction errors. It provides a simple form of feedback control that is assumed to be effective when there are no complicated dynamics. This mechanism is often compared to reflexes in biological and robot motor systems [1], [11], [23]. Figure 2.1 illustrates how an agent uses these updates to adapt its internal states and generate actions to minimize prediction errors.

2.4.3 Hierarchical Message Passing

As has been discussed before, the agents internal states are updated based on the gradient of the Free Energy with respect to those states. In the hierarchical case there are two sets of states to update: the mean of the dynamic states $\tilde{\mu}$ and the causal states v. The gradient of the Free Energy describes how the Free Energy changes with (small) changes in the states, and therefore how the states should be changed in order to minimize the Free Energy.

The gradient of the Free Energy can be expressed in matrix form, as was previously shown in 2.32. Here the terms ε_ω and ε_z are vectors that contain all the prediction errors for the functions $f(\tilde{\mu}, v)$ and $g(\tilde{\mu}, v)$ respectively. This also means that the partial derivatives $\frac{\delta\varepsilon}{\delta\tilde{\mu}}$ and $\frac{\delta\varepsilon}{\delta\tilde{v}}$ are Jacobians.

$$\varepsilon_\omega = D\tilde{\mu} - f(\tilde{\mu}, v) \tag{2.35}$$
$$\varepsilon_z = v - g(\tilde{\mu}, v) \tag{2.36}$$

$$\frac{\delta F}{\delta \mu} = \frac{\delta \varepsilon_\omega}{\delta \mu} \cdot \Pi_\omega \cdot \varepsilon_\omega + \frac{\delta \varepsilon_z}{\delta \mu} \cdot \Pi_z \cdot \varepsilon_z \tag{2.37}$$
$$\frac{\delta F}{\delta v} = \frac{\delta \varepsilon_\omega}{\delta v} \cdot \Pi_\omega \cdot \varepsilon_\omega + \frac{\delta \varepsilon_z}{\delta v} \cdot \Pi_z \cdot \varepsilon_z \tag{2.38}$$

It is clear that the gradient of the Free Energy with respect to a certain state only depends on the prediction error that include that state. If we consider the prediction errors in a hierarchy 2.22, we can see that given the states $\tilde{\mu}_{[m]}, v_{[m]}$ in a specific hierarchical level m:

$$\begin{aligned} \varepsilon_{\omega_{[m]}} &= D\tilde{\mu}_{[m]} - f(\tilde{\mu}_{[m]}, v_{[m]}) \\ \varepsilon_{z_{[m]}} &= v_{[m-1]} - g(\tilde{\mu}_{[m]}, v_{[m]}) \end{aligned} \tag{2.39}$$

$$\frac{\delta F}{\delta \tilde{\mu}_{[m]}} = \frac{\delta \varepsilon_{\omega[m]}}{\delta \tilde{\mu}_{[m]}} \cdot \Pi_{\omega[m]} \cdot \varepsilon_{\omega_{[m]}} + \frac{\delta \varepsilon_{z_{[m]}}}{\delta \tilde{\mu}_{[m]}} \cdot \Pi_{z_{[m]}} \cdot \varepsilon_{z_{[m]}} \tag{2.40}$$
$$\frac{\delta F}{\delta v_{[m]}} = \frac{\delta \varepsilon_{\omega_{[m]}}}{\delta v_{[m]}} \cdot \Pi_{\omega[m]} \cdot \varepsilon_{\omega_{[m]}} + \frac{\delta \varepsilon_{z_{[m]}}}{\delta v_{[m]}} \cdot \Pi_{z_{[m]}} \cdot \varepsilon_{z_{[m]}} + \frac{\delta \varepsilon_{z_{[m+1]}}}{\delta v_{[m]}} \cdot \Pi_{z_{[m+1]}} \cdot \varepsilon_{z_{[m+1]}} \tag{2.41}$$

If we consider these gradients carefully, it becomes clear that the gradients with respect to the dynamic states $\mu_{[m]}$ depend on $\mu_{[m]}$, $v_{[m]}$ and $v_{[m-1]}$. The gradient with respect to the causal states $v_{[m]}$ depend on $\mu_{[m]}$, $v_{[m]}$, $v_{[m-1]}$, $\mu_{[m+1]}$ and $v_{[m+1]}$. The gradients are used to update the states, so these dependencies furnish signals that pass between the hierarchies. It is thought that the signals in the biological brain can be described in a similar manner [10]. Figure 2.2 shows how these states are represented and updated in the internal model of the agent.

As before in equations 2.34 and 2.32, the gradients are used to provide update rules for the internal states and the actions :

$$\dot{\tilde{\mu}} = D\tilde{\mu} - \kappa_\mu \cdot \frac{\delta F}{\delta \tilde{\mu}} \tag{2.42}$$

$$\dot{v} = -\kappa_v \cdot \frac{\delta F}{\delta v} \tag{2.43}$$

$$\dot{a} = -\kappa_a \cdot \frac{\delta s}{\delta a} \cdot \frac{\delta F}{\delta s} \tag{2.44}$$

The state updates are weighted by the learning rates $\kappa_\mu, \kappa_v, \kappa_\mu$ which determine the speed with which the states are adjusted based on the prediction errors. Note that the dynamic states $\tilde{\mu}$ are updated based on the states of their derivatives as well as the prediction errors. This is because they have a dynamic representation in the model [4]. These update steps represent the core of the algorithm that is used for the simulations in the next chapters.

[4]This is not done for the causal states v because we have disregarded the generalized coordinates for the causal states in this thesis. Therefore $Dv = 0$.

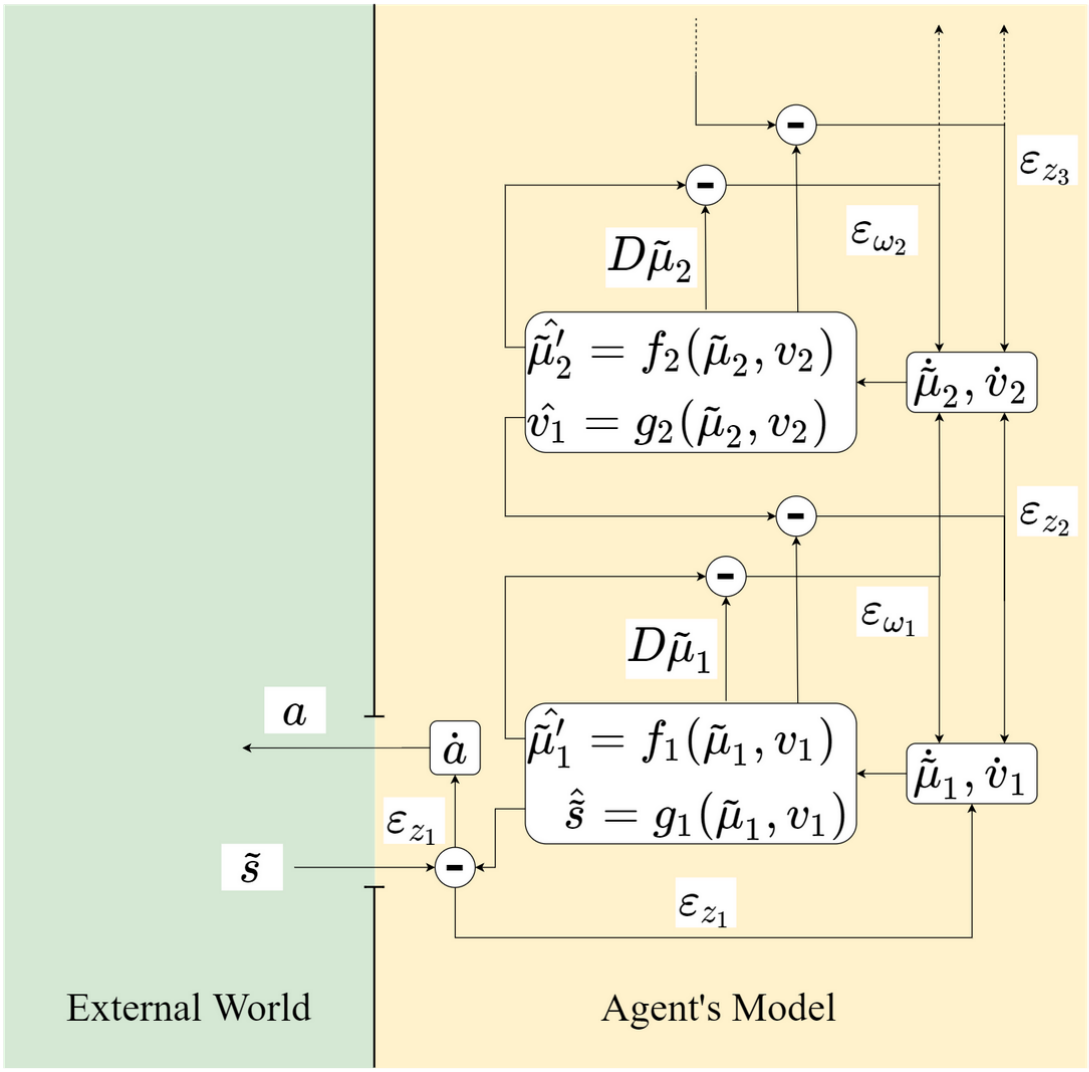


Figure 2.2: The agent still only interacts with the world through $\tilde{s}$ and a as in figure 2.1. In this case the internal model of the agent has been cast in a hierarchical form. The agent now assumes that there are causal states v that play a role in the dynamics producing the sensory observations $\tilde{s}$, and that these causal states themselves are generated by other dynamics, which are expressed in a higher hierarchical level. The states $\tilde{\mu}$ and v are updated based on the errors of the models predictions, and this can be interpreted as signals that move up and down the hierarchy.

Chapter 3

Simulation

This chapter gives a description of how hierarchical active inference can be simulated computationally. It explains how the concepts that have been explained in previous chapters are implemented in programming. The program described here is the result of the following research goal:

> Subgoal 3: Put a hierarchical active-inference based control algorithm into executable code for simulations, that can implement various internal and external models.

The code that is presented here has been written in `python`. The program is made up of several modules that each handle an aspect of the active inference algorithm. This information is meant to clarify the operation of the code that has been used in this thesis. The program is capable of handling various systems with functions for the physical dynamics and models for the beliefs of the agent. The agents model can be divided into hierarchical levels, which is the focus of this thesis. The code has been split into several sections, or modules that are explained in this chapter one by one. Figure 3.1 illustrates how these modules interact to build the models and run the simulation. Appendix A includes the code that was used to run the simulation for the hierarchical cart group system in chapter 5. The code was largely the same for the different simulations that are discussed in this thesis, and the main differences are present in the physics module A.2, in the models module A.4.2, and in the graphics module. The graphics code is extensive and not important for the purpose of explaining the simulations and so is not included in this report.

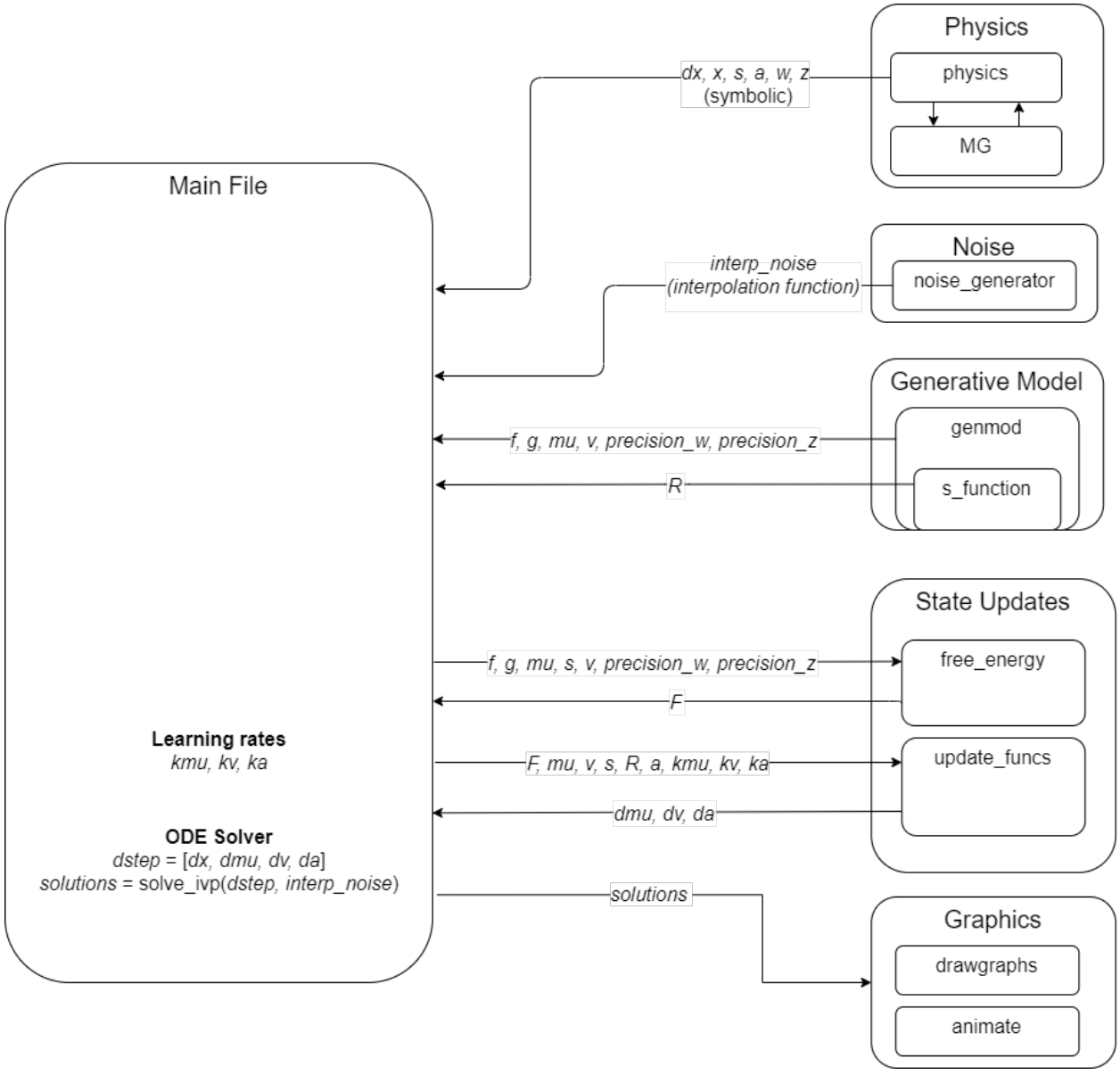


Figure 3.1: Flowchart showing the different processes involved in running the simulation. The arrows show the most important data that is passed between the modules

3.1 Variables and vectors

This table shows the notation that is used for all simulations throughout this thesis. Note that variables written in bold represent vectors or matrices, instead of single values.

Variable	
$\boldsymbol{x}, \dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}}$	real-world states and their derivatives
$\boldsymbol{s}$	sensory states
$\boldsymbol{a}$	actions
$\boldsymbol{\omega}$	dynamic noise acting on the states $\boldsymbol{x}$
$\boldsymbol{z}$	observation noise acting on sensory states $\boldsymbol{s}$
$\boldsymbol{\mu}$	agents internal beliefs
$\boldsymbol{v}$	causal beliefs
$\boldsymbol{f_p}$	transitive function of the physical process
$\boldsymbol{g_p}$	generative function of the physical process
$\boldsymbol{f}$	transitive function of the generative model
$\boldsymbol{g}$	generative function of the generative model
$\boldsymbol{R}$	Internal belief of action-senses relation, $\frac{\delta \boldsymbol{s}}{\delta \boldsymbol{a}}$
$\boldsymbol{\Pi}_\omega$	Precisions on dynamic errors
$\boldsymbol{\Pi}_z$	Precisions on sensory/causal state errors
k_μ	Internal belief update learning rate
k_v	Causal belief update learning rate
k_a	Action update learning rate
F	Free Energy

3.2 Generative process & Generative model

In this thesis, the simulation of active inference processes are is done through the numerical integration of a set of states, $\mathbf{x}$, $\mathbf{y}$, $\boldsymbol{\mu}$, $\boldsymbol{v}$ and $\mathbf{a}$. The dynamic states ($\mathbf{x}$), the observable states $\mathbf{y}$ and actions $\mathbf{a}$ belong to the simulated external world. They represent the physical development of the agents environment. The internal beliefs $\boldsymbol{\mu}$ and the causal states $\boldsymbol{v}$ belong to the simulated internal system of the agent. They represent the ideas and motivations that the agent controls.

This distinction between external and internal is also made in the functions that describe their change in time, the generative process and the generative model respectively.

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f_p}(\boldsymbol{x}, \boldsymbol{a}) + \boldsymbol{\omega} \\ \boldsymbol{y} &= \boldsymbol{g_p}(\boldsymbol{x}) + \boldsymbol{z} \end{aligned} \tag{3.1}$$

$$\begin{aligned} \hat{\boldsymbol{\mu}'} &= \boldsymbol{f}(\boldsymbol{\mu}, \boldsymbol{v}) \\ \hat{\boldsymbol{s}} &= \boldsymbol{g}(\boldsymbol{\mu}, \boldsymbol{v}) \end{aligned} \tag{3.2}$$

While the generative process described by f_p and g_p directly describe the development of the external states, the generative model (f, g) does not directly describe the development of the internal states, but rather predictions on the internal states. The actual change of the internal states in time depends on the update rules that are defined through gradient descent on the free energy. These update rules are defined by $\dot{\boldsymbol{\mu}}$, $\dot{\boldsymbol{v}}$ and $\dot{\boldsymbol{a}}$. These, and the time derivative of the real world states $\dot{\boldsymbol{x}}$ are then numerically integrated using a Runge-Kutta 45 solving method.
The simulation process is written in `python` and makes heavy use of the `sympy` toolbox to declare

symbolic variables. This allows the construction of expressions without assigning actual values to the variables, and allows certain operations such as calculating the partial derivatives of those functions. This is an important functionality when deriving the gradient descent on the Free-Energy expression.

3.2.1 Physics

The generative process is defined in the `physics` module of the program (Appendix A.2). Here the states $\boldsymbol{x}$ and the actions $\boldsymbol{a}$ are declared symbolically and the functions $\boldsymbol{f_p}$ are defined. $\boldsymbol{\omega}$ and $\boldsymbol{z}$ are declared as vectors of symbols that represent the noise terms. For the sake of simplicity, the states are assumed to be directly observable and so the sensory states $\boldsymbol{s}$ are simply the states $\boldsymbol{x}$ plus observation noise $\boldsymbol{z}$. Therefore, the functions $\boldsymbol{g_p}$ are not defined.

For the robot arm, the differential equations are relatively complicated. The system is described in rotational coordinates and includes things like coriolis and centrifugal forces, and must take into account the force interaction between the two segments. In a separate module `robotarmMg` the matrices for the differential equations are calculated using the TMT method [25] using the known constraints and forces on the robot arm. This module is not included in the appendix.

3.2.2 Generative model

The generative model is defined in the `generative_model` module of the program (A.4). This part declares the internal states $\boldsymbol{\mu}$ and the causal states $\boldsymbol{v}$ in symbolic form. Expressions are given for the generative and transitive functions $\boldsymbol{f}$ and $\boldsymbol{g}$. The precision matrices Π_ω and Π_z are also specified. These elements are taken from the module `models` (A.4.2) where the hierarchical levels are defined separately.

The generative model can consist of multiple hierarchical levels i, with a number of internal beliefs n that can have representations of derivatives up to a dynamical order m. All these different states are stacked together in a single vector in a particular order. First the different states, then the derivatives of all those states, and subsequently the states and their derivatives of higher hierarchical levels.

An important element when dealing with generalized coordinates is the derivative operator $\boldsymbol{D}$. The derivative operator is a matrix that shifts the elements in $\boldsymbol{\mu}$ in such a way that it yields the derivative $\boldsymbol{\mu'}$. The $\boldsymbol{D}$ matrix causes the elements of the highest dynamical order m to be replaced with zeroes, as there are no derivatives of those to select. 3.3 shows an example with the internal beliefs of two states, x_a and x_b and their derivatives. In `generative_model` (A.4.1 line 26-30), the appropriate $\boldsymbol{D}$ is constructed based on the number of states (excluding higher orders) that is

obtained from the models.

$$\boldsymbol{\mu_x} = \boldsymbol{D} \cdot \boldsymbol{\mu} \qquad \boldsymbol{\mu_{x'}} = \begin{bmatrix} \mu_{x'_a} \\ \mu_{x''_a} \\ 0 \\ \hline \mu_{x''_b} \\ \mu_{x'_b} \\ 0 \end{bmatrix} = \left[\begin{array}{ccc|ccc} 0 & 1 & 0 & & & \\ 0 & 0 & 1 & & & \\ 0 & 0 & 0 & & & \\ \hline & & & 0 & 1 & 0 \\ & & & 0 & 0 & 1 \\ & & & 0 & 0 & 0 \end{array}\right] \cdot \begin{bmatrix} \mu_{x_a} \\ \mu_{x'_a} \\ \mu_{x''_a} \\ \hline \mu_{x_b} \\ \mu_{x'_b} \\ \mu_{x''_b} \end{bmatrix} \tag{3.3}$$

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_{1,1,1} \\ \mu_{1,1,2} \\ \vdots \\ \mu_{1,1,n} \\ \mu_{1,2,1} \\ \vdots \\ \mu_{1,m,n} \\ \mu_{2,1,1} \\ \vdots \\ \mu_{i,m,n} \end{bmatrix} \qquad \boldsymbol{\mu'} = \boldsymbol{D} \cdot \boldsymbol{\mu} = \begin{bmatrix} \mu_{1,2,1} \\ \mu_{1,2,2} \\ \vdots \\ \mu_{1,2,n} \\ \mu_{1,3,1} \\ \vdots \\ 0 \\ \mu_{2,2,1} \\ \vdots \\ 0 \end{bmatrix} \tag{3.4}$$

3.2.3 Noise

Although the noises in this thesis are always set to 0, the program allows for the addition of dynamic noise $\boldsymbol{\omega}$ and observation noise $\boldsymbol{z}$, although this was not used in the simulations in this thesis. These noise terms are generated in the module `noise_generator`. It facilitates the creation of gaussian smooth noise by generating a random value for each time step according to a gaussian distribution, and then performing a convolution over this sequence with a gaussian kernel, which ensures a time-correlation between subsequent values. This noise sequence is then transformed into an interpolation function. The reason for this is that the integration algorithm may need to evaluate values of the noise that are not included in the (discrete) noise sequence. It should then be able to interpolate between the noise values.

Although this functionality exists in the code, it is not used in this thesis. It is still unclear how noise should exactly be implemented in this simulation. An unresolved issue is whether all the derivatives of the noise should be modelled, or only a disturbance in the form of an applied force, for example, which would be more accurate to real-world physics. For this reason, and because handling of noise is not a priority in this thesis, the noise implementation is excluded from the simulations and is left for future work. This means that for all simulations discussed in this report both the noises $\boldsymbol{\omega}$ and $\boldsymbol{z}$ are zero.

3.3 State Updates

The `state_updates` (A.5) part of the program is responsible for defining the update expressions and executing the numerical integration. The update expressions $\dot{\boldsymbol{\mu}}$, $\dot{\boldsymbol{v}}$ and $\dot{\boldsymbol{a}}$ are derived from the previously defined elements and integrated numerically.

3.3.1 Free energy

The function `free_energy` (A.5.2) derives an expression for the free energy F based on the given $\boldsymbol{f}$, $\boldsymbol{g}$, $\boldsymbol{\mu}$, $\boldsymbol{v}$, and $\boldsymbol{s}$.

$$\begin{aligned} \varepsilon_\omega &= \boldsymbol{D_\mu} \cdot \boldsymbol{\mu} - \boldsymbol{f} & \varepsilon &= \begin{bmatrix} \varepsilon_\omega \\ \varepsilon_z \end{bmatrix} \\ \varepsilon_z &= \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{v} \end{bmatrix} - \boldsymbol{g} & \boldsymbol{\Pi} &= \begin{bmatrix} \boldsymbol{\Pi_\omega} & \\ & \boldsymbol{\Pi_z} \end{bmatrix} \end{aligned} \tag{3.5}$$

$$F = \frac{1}{2} \cdot \varepsilon^T \cdot \boldsymbol{\Pi} \cdot \varepsilon \tag{3.6}$$

3.3.2 Update Functions

The `update_funcs` (A.5.3) module contains the functions `mu_update`, `v_update` and `a_update`. Each of these takes the jacobian (denoted by $\frac{\delta F}{\delta \boldsymbol{\mu}}$) of the Free-Energy F with respect to the appropriate vector and uses it to determine how the respective states should change in the next update step. The `a_update` function uses $\boldsymbol{R}$ which is the variable `s_function` in the code. This is a vector that describes the elements of $\boldsymbol{s}$ as a function of $\boldsymbol{a}$ which is passed from the `solver` module.

`mu_update`

$$\dot{\boldsymbol{\mu}} = \boldsymbol{D} \cdot \boldsymbol{\mu} - \kappa_\mu \cdot \left(\frac{\delta F}{\delta \boldsymbol{\mu}} \right) \tag{3.7}$$

`v_update`

$$\dot{\boldsymbol{v}} = -\kappa_v \cdot \left(\frac{\delta F}{\delta \boldsymbol{v}} \right) \tag{3.8}$$

`a_update`

$$\dot{\boldsymbol{a}} = -\kappa_a \cdot \left(\frac{\delta F}{\delta \boldsymbol{s}} \cdot \frac{\delta \boldsymbol{R}}{\delta \boldsymbol{a}} \right) \tag{3.9}$$

Some of the update functions contain the sensory states $\boldsymbol{s}$. These terms are replaced with the real-world states plus the observation noise components ($\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{z}$).

3.3.3 Integration

Once all the update steps are defined, they can be put into one large vector which is passed to the solver. This happens in `solve` (A.5.1). `solve_ivp` from `scipy.integrate` is used, which is an initial value problem solver that can integrate a set of differential equations given some initial values. For the cases in this report, `solve_ivp` is set to a Runge-Kutta 45 solver, with a relative tolerance of 10^{-4} and an absolute tolerance of 10^{-7}.

$$initial\_values = \begin{bmatrix} \boldsymbol{x_0} \\ \boldsymbol{\mu_0} \\ \boldsymbol{v_0} \\ \boldsymbol{a_0} \end{bmatrix} \qquad dstep = \begin{bmatrix} \dot{\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{z}) \\ \dot{\boldsymbol{\mu}}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{z}) \\ \dot{\boldsymbol{v}}(\boldsymbol{\mu}, \boldsymbol{v}, \boldsymbol{z}) \\ \dot{\boldsymbol{a}}(\boldsymbol{x}, \boldsymbol{\mu}, \boldsymbol{z}) \end{bmatrix} \tag{3.10}$$

We can see that the differential equations of the states are functions of the states themselves and of the noise terms z. This noise is generated beforehand and passed to the solver as an interpolation function `interp_noise(t)`. This means that during the integration, the solver can determine the appropriate noise terms at time t. The solver carries out the integration and returns the values for all states at each time step.

```
solution = solve_ivp(dstep,initial_values,interp_noise)
```

3.4 Graphics

To be able to intuitively asses the behaviour of the results that the code produces, the time-series solutions for the states are represented graphically. The solutions generated by `solve_ivp` are passed to the `graphics` module. This module contains `drawgraphs` functions that produce line graphs of the states, and `animation` functions that produce animations of the simulated system using the `matplotlib` library. These are custom functions that have to be remade for different generative processes and models, to make sense of the solutions. Due to the length and variety of the models this code is not included in the appendix.

3.5 Chapter Summary

To summarize, this chapter has described the body of code that has been used in this thesis to simulate the active-inference based control method. It provides the basis for all the simulations that are presented in this report. Various physical systems can be simulated in the `physics` (A.2) module, and the `generative_model` module allows for the implementation of arbitrary hierarchical models. After the physical and generative models have been defined, the main parameters that can be tweaked are the learning rates κ in the main file and the precision matrices Π which are specified in the `models` (A.4.2) module in `generative_model`.

The fact that the code is model-independent makes it very versatile. However, implementing a new physical and/or internal model requires adapting the `physics` and `models` part of the code. Additionally, to obtain meaningful results such as graphs and animations from these simulations means adapting the graph and animation generating functions in `graphics`.

Chapter 4

Single Cart

The previous chapter explained how the principles of active inference can be cast into an algorithm that generates and evaluates predictions. The algorithm can be used as a control method, which is how it is approached in this thesis. It may not be clear yet how this control is achieved in practice, and this leads us to the next research goal:

> Subgoal 4: Demonstrate the algorithm for a single-level linear goal-reaching control problem.

This thesis is focused specifically on applying this method in a hierarchical context, but first the basic operation of this algorithm with a single-level model will be shown. By applying the method to a very simple 1-dimensional control problem, some of the issues and complexities of this method can be highlighted. Section 4.2 demonstrates the implementation of a first order simple cart model. An analysis is provided of the stability of the controlled system, and how this is affected by the many parameters in the generative model.

Additionally, the effect of generalized coordinates will be investigated. These representations of multiple dynamical orders are an important part of the active inference theory, but it is unclear if they contribute much to the performance of this simple noiseless system. Section 4.4 discusses the same simulation but with the addition of second order beliefs, to observe the effect of embedding order. The difference in results between the 1st and 2nd order implementation turn out to be negligible. Therefore, for the scope of this thesis, internal states of a higher order than the sensory states will be omitted.

This chapter considers the generative process and generative model for a single 1-D cart on a horizontal plane. The cart is modelled by a point mass that can move in two directions (left and right) and is influenced by a damping force b and a force a exerted by the agent. The generative model, however, assumes that the cart is not affected by damping or other forces and simply presumes that it moves towards a given goal with first-order dynamics. The action a then mitigates this difference by moving the physical cart towards the goal. The difference between the generative model and the physical system is the key that generates the control actions.

4.1 Physical model

The first part of the simulation that is specified is the physical process, also called the generative process or "external world" in relation to the agent. For the single cart system it is described by the differential equations 4.1 and 4.2. As this is a linear system, it can be cast into a state-space representation.

$$\dot{\boldsymbol{x}} = A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u} \qquad (4.1)$$

$$\boldsymbol{s} = C \cdot \boldsymbol{x} \qquad (4.2)$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{M} \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \quad \boldsymbol{s} = \begin{bmatrix} s \\ \dot{s} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} a \end{bmatrix}$$

$x, \dot{x}, \ddot{x}$: Position, velocity and acceleration
b : Damping constant
M : Mass
$s, \dot{s}$: sensory observations[1]of position and velocity
a : action, in this case a force on the cart

Note that the sensory information here (4.2) is simply the position and acceleration, meaning the states are directly observable. Usually a process would include noise terms $\boldsymbol{\omega}$ and $\boldsymbol{z}$ for $\boldsymbol{x}$ and $\boldsymbol{s}$ respectively, but these are not considered here for simplicity.

4.2 Single Cart Simulation

The internal states of the agent and their updates are also included in the simulations as a set of differential equations. These updates are done based on the generative model. This is the model that the agent employs to make its predictions. In this particular implementation, the generative model describes the same point mass as the physical model but instead of modelling forces on it, the cart is assumed to move towards a goal, with a velocity proportional to the distance from that goal. This is similar to the approach in [22] where it is applied to a robot arm.

An important point that is demonstrated here is that the generative model does not necessarily need reflect the conditions of the outside world. Rather, the agent (an organism or robot) can possess a belief of desired states which it will try to achieve through action. Therefore the belief of the agent eventually determines how the states in the real world will develop (if this within the realm of possibility). In this single cart simulation the choice has been made to model goal-reaching behaviour. This offers a clear criterion to evaluate the performance of the algorithm.

[1]In control theory, the conventional symbol for observations is y. Here, the symbol s is used to specify sensory observations, maintaining the analogy of a biological agent.

This behaviour is described in the simple differential equations shown below in section 4.2.1(4.3,4.4). To run the simulation and obtain the results, the state updates rules are used as given in 2.44 in chapter 2. These update rules are the differential equations that are integrated in the simulation, together with the physical equations 4.1 and 4.2.

4.2.1 Generative Model

Below are the generative functions $\boldsymbol{f}$ and $\boldsymbol{g}$ for a first order model. This means only variables and expressions up to the velocity $\mu_{x'}$ are included. In addition, there is an expression $\boldsymbol{R}$, which describes the belief the agent has about how sensory information is produced by the actions a. In this case the agent assumes that both the position and velocity signals from the cart are directly proportional to the position and the velocity of the cart. This is not accurate of course, as the action force actually affects the acceleration of the cart. However, the acceleration is not available as a sensory signal. In practice, the assumed $\boldsymbol{R}$ allows the agent to exert a form of control based on the sensory prediction error on position and velocity.

The precision matrices Π represents the accuracy with which the agent expects to be able to determine the internal states and greatly affects how aggressively the states are updated. Generally, the values for the precisions are related to the assumed variances of the noise terms. As noise is disregarded in this thesis, there are no noise parameters to base the precisions on in this case. This means that the values for Π are free to be chosen, and are available for tuning the system. An additional assumption that is made is that the precision matrices Π only consist of diagonal terms, indicating that the prediction errors on the dynamical orders of the same state are not correlated. The converging parameter c determines the velocity at which the cart is assumed to move towards the goal.

$$\boldsymbol{f} = \begin{bmatrix} \hat{\dot{\mu}}_x \\ \hat{\mu}_{x'} \end{bmatrix} = \begin{bmatrix} -c & 0 \\ 0 & -c \end{bmatrix} \cdot \begin{bmatrix} \mu_x \\ \mu_{x'} \end{bmatrix} + \begin{bmatrix} c \cdot \eta_g \\ 0 \end{bmatrix} \quad (4.3)$$

$$\boldsymbol{g} = \begin{bmatrix} \hat{s} \\ \hat{\dot{s}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mu_x \\ \mu_{x'} \end{bmatrix} \quad (4.4)$$

$$\boldsymbol{R} = \begin{bmatrix} \hat{s} \\ \hat{\dot{s}} \end{bmatrix} = \begin{bmatrix} a \\ a \end{bmatrix}$$

$$\Pi_\omega = \begin{bmatrix} \pi_{\omega_1} & 0 \\ 0 & \pi_{\omega_2} \end{bmatrix} \quad (4.5)$$

$$\Pi_z = \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \quad (4.6)$$

c : Converging parameter
$\mu_x, \mu_{x'}$: Belief of x position and the belief of the first derivative x'
$\hat{s}, \hat{\dot{s}}$: Estimated sensory observations
η_g : Goal position
Π_ω : Precision matrix on dynamics
Π_z : Precision matrix on observations

Note that there are two expressions that represent predictions of the sensory signals $(s, \dot{s})$. $\boldsymbol{g}$ makes predictions based on the internal states μ_x, μ'_x, and $\boldsymbol{R}$ predicts how the action a affects the signals.

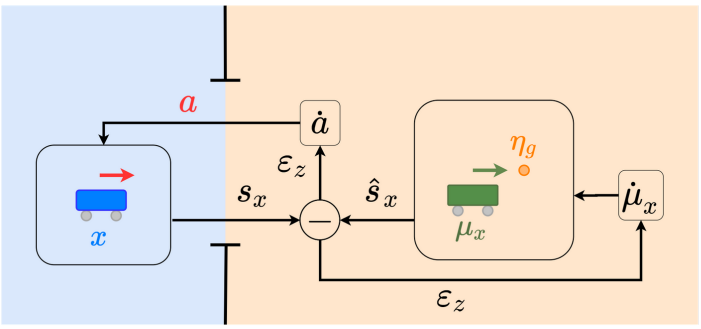


Figure 4.1: This diagram provides an overview of the interaction between the external (physical) world and the internal model maintained by the agent. The agents model has a belief of the position of the cart μ_x, and uses this to make predictions about the sensory information s. The resulting prediction error is used to update actions $\dot{a}$ and internal beliefs $\dot{\mu}_x$.

4.2.2 State updates

Shown below is the derivation of the updates for the internal beliefs $\boldsymbol{\mu}$ for the 1st order model. These update steps have been explained in 2.44. Combining this with the physical equations (4.1,4.2) produces the closed loop response 4.11. This can be used to determine the stability of the system.

$$\frac{\delta \boldsymbol{f}}{\delta \boldsymbol{\mu}} = \begin{bmatrix} -c & 0 \\ 0 & -c \end{bmatrix} \qquad (4.7) \qquad \frac{\delta \boldsymbol{R}}{\delta \boldsymbol{a}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\frac{\delta \boldsymbol{g}}{\delta \boldsymbol{\mu}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (4.8) \qquad D = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\begin{aligned} \dot{\boldsymbol{\mu}} &= D\mu - \kappa_\mu \cdot \left[\left(D - \frac{\delta f}{\delta \mu} \right)^T \cdot \Pi_\omega \cdot (D\mu - f(\mu)) - \frac{\delta g(\mu)}{\delta \mu} \cdot \Pi_z \cdot (s - g(\mu)) \right] \\ &= \begin{bmatrix} \mu_{x'} \\ 0 \end{bmatrix} - \kappa_\mu \cdot \left[\begin{bmatrix} c & 1 \\ 0 & c \end{bmatrix}^T \cdot \begin{bmatrix} \pi_{\omega_1} & 0 \\ 0 & \pi_{\omega_2} \end{bmatrix} \cdot \begin{bmatrix} \mu_{x'} - c(\eta_g - \mu_x) \\ c\mu_{x'} \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \cdot \begin{bmatrix} s - \mu_x \\ s' - \mu_{x'} \end{bmatrix} \right] \end{aligned} \qquad (4.9)$$

$$\dot{\boldsymbol{a}} = -\kappa_a \cdot \frac{\delta F}{\delta \boldsymbol{s}} \cdot \frac{\delta R}{\delta \boldsymbol{a}} = -\kappa_a \cdot \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \begin{bmatrix} s - \mu_x \\ \dot{s} - \mu_{x'} \end{bmatrix} \right) \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \qquad (4.10)$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\mu}_x \\ \dot{\mu}_{x'} \\ \dot{a} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -\frac{b}{M} & 0 & 0 & \frac{1}{M} \\ \kappa_\mu \cdot \pi_{z_1} & 0 & -\kappa_\mu \cdot (c^2 \cdot \pi_{\omega_1} + \pi_{z_1}) & -c \cdot \kappa_\mu \cdot \pi_{\omega_1} + 1 & 0 \\ 0 & \kappa_\mu \cdot \pi_{z_2} & -c \cdot \kappa_\mu \cdot \pi_{\omega_1} & -\kappa_\mu \cdot (c^2 \cdot \pi_{\omega_2} + \pi_{\omega_1} + \pi_{z_2}) & 0 \\ -\kappa_a \cdot \pi_{z_1} & -\kappa_a \cdot \pi_{z_2} & \kappa_a \cdot \pi_{z_1} & \kappa_a \cdot \pi_{z_2} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ \dot{x} \\ \mu_x \\ \mu_{x'} \\ a \end{bmatrix} + \begin{bmatrix} \kappa_\mu \cdot (c^2 * \eta_{goal} \cdot \pi_{\omega_1}) \\ \kappa_\mu \cdot (c * \eta_{goal} \cdot \pi_{\omega_1}) \\ 0 \end{bmatrix} \qquad (4.11)$$

4.3 Stability

One issue that was encountered when attempting to run the active inference program on the 1-dimensional cart control problem was the fact that in certain situations the solution was unstable and the computational solver would produce very large values for the states. This instability could be related to the fact that the generative model differs considerably from the real-world system. There are assumptions about the dynamics that are not reflected in the real world which might lead to mistakes in the updates of the actions and internal beliefs. An example of such an assumption is the matrix R which assumes that the action a has a one-on-one effect on the position and the velocity.

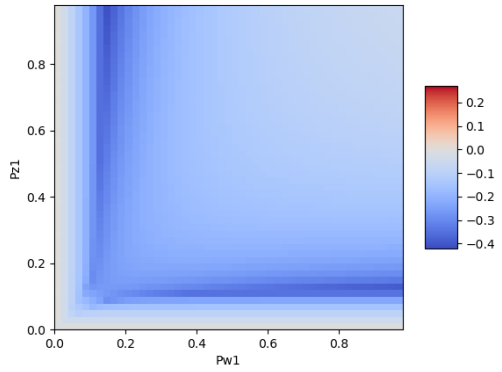
As this system is a set of linear differential equations, the stability can be analysed by observing the eigenvalues of the closed-loop matrix that describes the differential equations. This matrix (4.11) describes the development of the system as a whole, combining the generative process and the internal model. Note that this does not include constant terms (such as the goal in this case) or external factors(such as noise if it were included). The system is said to be stable if the real parts of all the eigenvalues of that matrix are negative. See *Grimbergen(2019)* ([15]) for a deeper analysis of active inference in state space form and stability.

Below a plot is given (4.2) of the maximum real eigenvalues for the system plotted against different values for various parameters as an example of the sensitivity of the system for changes in the parameters. An important thing to remember is that the eigenvalues provide information about the continuous system described by the differential equations. This means that any instabilities due to numerical inaccuracies of the integration process are not considered in this analysis and may still be an issue even if the real part of the eigenvalues are negative.

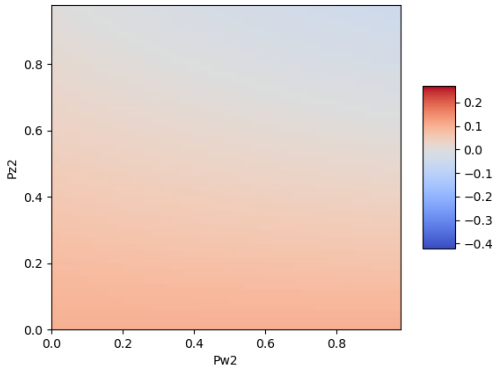
It should be noted that plots like figure 4.2 and 4.4 only provide information about stability for a very specific set of variables. If any of the parameters are changed, the maximum eigenvalue plot will change too. This method is an effective tool to determine parameters for a stable solution of the simulation through trial and error, but does not provide much insight into how the parameters influence the response of the system.

4.3.1 Results

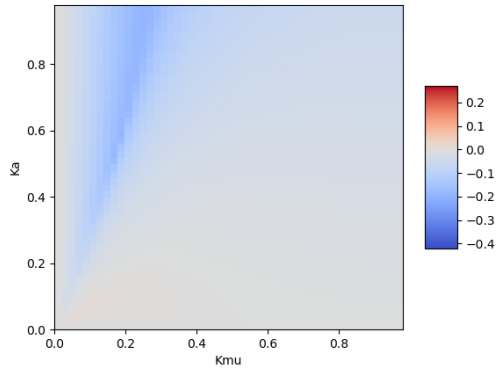
Using the information provided by figure 4.2, parameters can be chosen for a stable convergence of the single cart system. In this case only the value for π_{ω_2} was changed to 0.2 and all other parameters were left at a value of 1. Figure4.2a shows that this corresponds with more negative eigenvalues and this led to a reasonably quick and smooth convergence of the cart to the end goal at $\eta_{goal} = 3$. This demonstrates that the generative process can be effectively controlled towards an end goal by describing the necessary dynamics in the generative model with appropriate parameters. Now that this has been established, the next section will explore the effect of including higher order dynamics in the generative model.



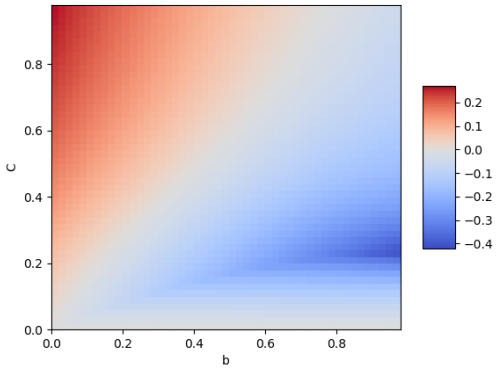
(a) Maximum eigenvalues for $\pi_{z_1}, \pi_{\omega_1}$



(b) Maximum eigenvalues for $\pi_{z_2}, \pi_{\omega_2}$



(c) Maximum eigenvalues for K_a, K_μ



(d) Maximum eigenvalues for c, b

Figure 4.2: The maximum of the real part of the eigenvalues of the differential equation matrix for the 1-D 1st order single cart system, plotted for different values for the various parameters. The system is stable when the maximum real part of the eigenvalues is less than 0. Each figure shows two of the parameters plotted against each other for values from 0 to 1 when all other parameters are 1. These plots illustrate that the choice of parameters greatly affects stability.

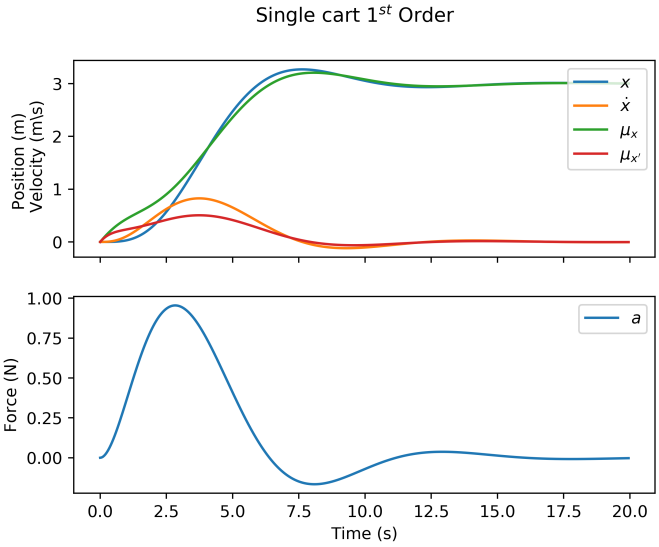


Figure 4.3: The states of the cart and the corresponding beliefs in the 1st order internal model. The initial position is at $x_0 = 0$ and the goal is $\eta_{goal} = 3$. Parameter values:($\Pi_{z1} = 1$, $\Pi_{\omega 1} = 0.2$, $\Pi_{z2} = 1$, $\Pi_{\omega 2} = 1$, $\kappa_\mu = 1$, $\kappa_a = 1$, $c = 1$, $b = 1$)

4.4 Including 2nd order states

The active inference based method involves generalized coordinates, and this means the generative model can include any number of dynamical orders. Including higher orders simply means that there are additional representations of higher derivatives and additional functions to predict them. The highest order represented in the model is also called its embedding order. Here we consider a model containing derivatives up to the 2nd order (position, velocity and acceleration). The results are compared to the previously discussed first order implementation to see if there is any difference in behaviour. Generalized coordinates in active inference are mainly a means to consider the dynamics of differentiable (smooth) noise in systems. As noise is disregarded in this case, the effect of including higher derivatives may be limited.

4.4.1 Generative model

This version of the generative model expands the state representation with one more derivative compared to 4.3. This model includes an extra state $\mu_{x''}$ which develops according to the assumed dynamics. Note that there are still only predictions for the sensory information of the position and velocity of the cart $(\hat{s}, \hat{\dot{s}})$. This also means that there are only two corresponding precisions for the sensory information (π_{z_1}, π_{z_2}).

$$\boldsymbol{f} = \begin{bmatrix} \hat{\dot{\mu}}_x \\ \hat{\dot{\mu}}_{x'} \\ \hat{\dot{\mu}}_{x''} \end{bmatrix} = \begin{bmatrix} -c & 0 & 0 \\ 0 & -c & 0 \\ 0 & 0 & -c \end{bmatrix} \cdot \begin{bmatrix} \mu_x \\ \mu_{x'} \\ \mu_{x''} \end{bmatrix} + \begin{bmatrix} c \cdot \eta_g \\ 0 \\ 0 \end{bmatrix} \tag{4.12}$$

$$\boldsymbol{g} = \begin{bmatrix} \hat{s} \\ \hat{\dot{s}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mu_x \\ \mu_{x'} \end{bmatrix} \tag{4.13}$$

$$\Pi_\omega = \begin{bmatrix} \pi_{\omega 1} & 0 & 0 \\ 0 & \pi_{\omega 2} & 0 \\ 0 & 0 & \pi_{\omega 3} \end{bmatrix} \tag{4.14}$$

$$\Pi_z = \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \tag{4.15}$$

4.4.2 State updates

Shown on the next page is the derivation of the update for the internal beliefs $\boldsymbol{\mu}$ for the 2nd order model. The update of the extra state $\mu_{x''}$ is influenced by the believed velocity $\mu_{x'}$, and vice versa, but there is no corresponding sensory state $\ddot{s}$ for feedback. This suggests that $\mu_{x''}$ simply acts as a stabilizing element to enforce the believed dynamics and that its relevance is limited in the absence of process noise. The closed loop response is excluded here for brevity.

$$\frac{\delta \boldsymbol{f}}{\delta \boldsymbol{\mu}} = \begin{bmatrix} -c & 0 & 0 \\ 0 & -c & 0 \\ 0 & 0 & -c \end{bmatrix} \qquad (4.16) \qquad \frac{\delta \boldsymbol{R}}{\delta \boldsymbol{s}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\frac{\delta \boldsymbol{g}}{\delta \boldsymbol{\mu}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (4.17) \qquad D = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{aligned} \dot{\boldsymbol{\mu}} =& D\mu - \kappa_\mu \cdot \left[\left(D - \frac{\delta f}{\delta \mu} \right)^T \cdot \Pi_\omega \cdot (D\mu - f(\mu)) - \frac{\delta g(\mu)}{\delta \mu} \cdot \Pi_z \cdot (s - g(\mu)) \right] \\ =& \begin{bmatrix} \mu_{x'} \\ \mu_{x''} \\ 0 \end{bmatrix} - \kappa_\mu \cdot \left[\begin{bmatrix} c & 1 & 0 \\ 0 & c & 1 \\ 0 & 0 & c \end{bmatrix}^T \cdot \begin{bmatrix} \pi_{\omega 1} & 0 & 0 \\ 0 & \pi_{\omega 2} & 0 \\ 0 & 0 & \pi_{\omega 3} \end{bmatrix} \cdot \begin{bmatrix} \mu_{x'} + c \cdot \mu_x - c \cdot \eta_g \\ \mu_{x''} + c\mu_{x'} \\ c\mu_{x''} \end{bmatrix} \right] \\ & - \kappa_\mu \left[- \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \cdot \begin{bmatrix} s - \mu_x \\ s' - \mu_{x'} \end{bmatrix} \right] \end{aligned} \qquad (4.18)$$

$$\dot{\boldsymbol{a}} = -\kappa_a \cdot \frac{\delta F}{\delta \boldsymbol{s}} \cdot \frac{\delta R}{\delta \boldsymbol{a}} = -\kappa_a \cdot \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \pi_{z_1} & 0 \\ 0 & \pi_{z_2} \end{bmatrix} \begin{bmatrix} s - \mu_x \\ \dot{s} - \mu_{x'} \end{bmatrix} \right) \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \qquad (4.19)$$

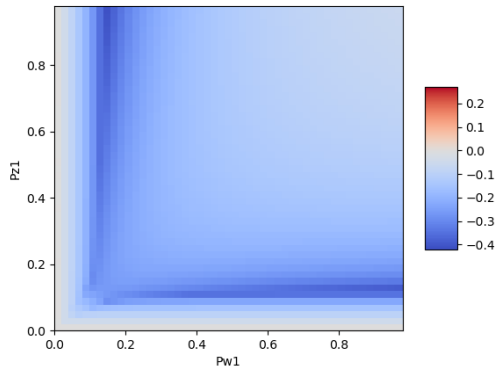
4.4.3 Stability

Just as was done for the 1st order system, the stability of the system with the 2nd order model was analysed. This was done by again calculating the maximum eigenvalues of the closed-loop matrix. This matrix has not been included here for brevity but it is a similar construction to that of the 1st order system (4.11). The eigenvalues were calculated for various values of the parameters and these were plotted in figure 4.4. The eigenvalue plots strongly resemble those of the first order system 4.2, especially 4.4a and 4.4b are almost identical. However, there are small differences when comparing figure 4.4c and 4.2c, or 4.4d and 4.2d. Apparently there are values for the parameters for which the 2nd order system is stable but the first order system is not. This means that including higher orders in the system does have some effect on the dynamics.

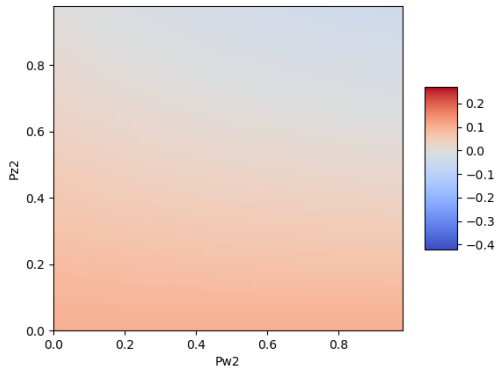
4.4.4 Results

The resulting response for the 2nd order model (figure 4.5) was very similar to the results of the 1st order system. In fact, the difference between the two responses was negligible. The stability plots in the previous section showed some differences in the dynamic properties for the 1st and 2nd order, but these differences in dynamics have not become apparent in the results presented here, because parameters were chosen for which both systems displayed similar (stable) behaviour.

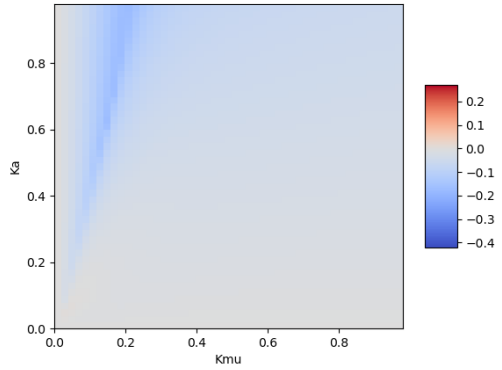
Overall, the higher dynamical order seems to have little effect on the dynamics. This might be be due to the fact that only the sensory information up to the 1st order was available to the agent. It must be noted that the method of analysing eigenvalues does not consider robustness in the presence of noise or disturbances. Higher dynamical orders could be more important if these are included, however this is outside the scope of this thesis, which is focused more on the method itself and hierarchical expansion of the models.



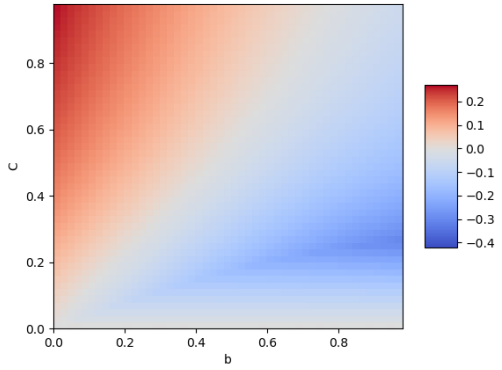
(a) Maximum eigenvalues for $\pi_{z_1}, \pi_{\omega_1}$



(b) Maximum eigenvalues for $\pi_{z_2}, \pi_{\omega_2}$



(c) Maximum eigenvalues for K_a, K_μ



(d) Maximum eigenvalues for c, b

Figure 4.4: The maximum of the real part of the eigenvalues of the differential equation matrix for the 1-D 2nd order single cart system, plotted for different values for the various parameters. The system is stable for values lower than 0. These plots show some minor differnces with the stability plots for the 1st order system.

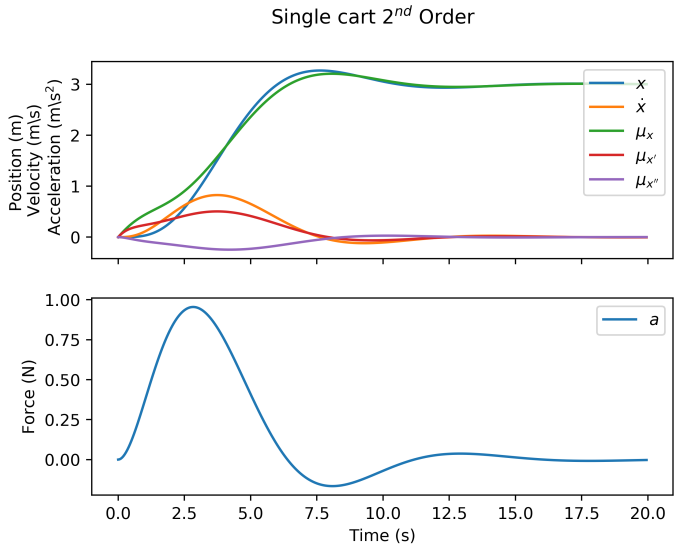


Figure 4.5: The states of the cart and the corresponding beliefs in the 2nd order internal model. Parameter values:($\Pi_{z1} = 1$, $\Pi_{\omega 1} = 0.2$, $\Pi_{z2} = 1$, $\Pi_{\omega 2} = 1$, $\kappa_\mu = 1$, $\kappa_a = 1$, $c = 1$, $b = 1$)

4.5 Chapter Summary

This chapter has demonstrated the algorithm for a very simple linear control problem. This provides a clear example of a generative process and a generative model and the dynamics that are simulated in this algorithm. These results demonstrate that the active inference based method that is applied in this thesis is capable of imposing a form of control.

It has also become clear that the active inference based algorithm is not necessarily stable, and that there are many parameters that influence the response of the system. In the case where there is no noise or noise parameters are not known, the precisions can be used as tuning parameters. If the generative process and the generative model are both linear, stability of the continuous system can be determined by analysing the eigenvalues of the system matrix. This allowed for an effective selection of parameters. However, this does require some analysis and is not a viable option for non-linear systems. Also, eigenvalue analysis does not provide any information about instabilities that may arise from the integration steps in the simulation. The application of active inference principles does not guarantee stability and parametrization is an important factor in establishing effective generative models for control purposes. This is an important point to consider in future implementations.

Another interesting observation that has been made in the implementation of this 1-dimensional cart control problem is that the inclusion of higher orders leads to very little difference in the resulting solution. At least in this linear process with 1st order sensor information and no noise, the observed responses are almost identical. This is the case for the particular (stable) parameters chosen in this system, but slight differences in the eigenvalue plots suggest that there are sets of parameters in which the higher orders have a significant influence. The relevance of higher dynamical orders may also change if noise is included in the generative process or if the physical system that is being controlled includes higher dynamics. For the scope of this thesis, it will be assumed that this is not the case and that states of higher derivatives than 1st order can be neglected. The focus of this thesis is the implementation of a hierarchical generative model, which will be the subject of the following chapters.

Chapter 5

Cart Groups

The previous chapter demonstrated the application of the active inference based algorithm with a single-level linear model. This provides a first glance at how this algorithm can be implemented in practice, albeit for a very simplified system. However, the main aim of this thesis is to explore the application and behaviour of hierarchical models within this method. What does the generative model look like when the agent assumes a hierarchical structure of the world? And how can goal-directed control behaviour be integrated into this approach? These questions will be answered with the second part of the algorithm demonstration:

> Subgoal 3: Demonstrate the algorithm for a linear goal reaching control problem using a hierarchical generative model.

In this chapter a group of 8 carts are simulated on a 1-dimensional line. The individual carts are influenced by the same dynamics as the cart in the previous chapter. They are affected by a damping force and an action force a exerted by the agent. The main difference in this implementation is the model employed by the agent. This model consists of multiple levels that are organized in a hierarchical fashion. Specifically, the agent models the 8 different carts and describes their movement towards a goal in groups of two. This means there are 4 different goals, and these are represented in a second level. These, in turn, move towards two goals represented in a third level, which move towards a fixed 'prior' goal. This hierarchy is not present in the actual physical world, but exists only in the agent's model. The simulation will show that this hierarchical behaviour can be realized in the physical system by the agent through the actions. In a more general sense, this means that an agent is able to divide a large goal or task (move all carts towards a goal position) into smaller sub-tasks (move two carts towards a goal-position).

As motivated in the previous chapter, noise and higher dynamical orders are omitted in these simulations. Finding the parameters that result in stable and smooth convergence towards a goal remains a process of trial and error. The parameters that have been used in this chapter differ from those in the previous chapter as the hierarchical structure changes the dynamics of the system slightly.

5.1 Physical model

The simulations of the physical 'real world' carts is done in the same way as in the previous chapter for the single cart simulation, with the same damping coefficient and action forces, only in this case there are 8 carts instead of 1.

$$\dot{\boldsymbol{x}} = A \cdot \boldsymbol{x} + B \cdot \boldsymbol{u} \tag{5.1}$$

$$\boldsymbol{s} = C \cdot \boldsymbol{x} \tag{5.2}$$

$$A = \begin{bmatrix} 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 \\ 0 & \dots & 0 & \frac{b}{M_1} & \dots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & \frac{b}{M_8} \end{bmatrix} \quad B = \begin{bmatrix} 0 & & & & & \\ & \ddots & & & & \\ & & 0 & & & \\ & & & \frac{1}{M} & & \\ & & & & \ddots & \\ & & & & & \frac{1}{M} \end{bmatrix} \quad C = \begin{bmatrix} 1 & \dots & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_8 \\ \ddot{x}_1 \\ \vdots \\ \ddot{x}_8 \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_8 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_8 \end{bmatrix} \quad \boldsymbol{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_8 \\ \dot{s}_1 \\ \vdots \\ \dot{s}_8 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a_1 \\ \vdots \\ a_8 \end{bmatrix}$$

$x_n, \dot{x}_n, \ddot{x}_n$: Position, velocity and acceleration of each cart
b : Damping constant
M : Mass, equal for each cart
$s_n, \dot{s}_n$: sensory observations of position and velocity for each cart
a_n : actions, forces on each cart

5.2 Generative model

In this simulation the generative model is structured in a hierarchical way. There are different ways to organize a model hierarchically. Hierarchy can become apparent in the relation between cause and effect, part and whole or goals and sub-goals. One of the suggested functionalities of a hierarchy is to represent goals that can be divided into different sub-goals or tasks. The hierarchical model described here follows that idea in a simplified way.

$$\boldsymbol{f_3} = \begin{bmatrix} -c & 0 & 0 & 0 \\ 0 & -c & 0 & 0 \\ 0 & 0 & -c & 0 \\ 0 & 0 & 0 & -c \end{bmatrix} \cdot \begin{bmatrix} \mu_{3e} \\ \mu_{3f} \\ \hline \mu'_{3e} \\ \mu'_{3f} \end{bmatrix} + \begin{bmatrix} c \cdot \eta_g \\ c \cdot \eta_g \\ \hline 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{g_3} = \begin{bmatrix} \mu_{3e} \\ \mu_{3f} \end{bmatrix} \tag{5.3}$$

$$\boldsymbol{f_2} = \begin{bmatrix} -c & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & -c \end{bmatrix} \cdot \begin{bmatrix} \mu_{2a} \\ \mu_{2b} \\ \mu_{2c} \\ \mu_{2d} \\ \hline \mu'_{2a} \\ \mu'_{2b} \\ \mu'_{2c} \\ \mu'_{2d} \end{bmatrix} + \begin{bmatrix} v_{2e} \\ v_{2e} \\ v_{2f} \\ v_{2f} \\ \hline 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{g_2} = \begin{bmatrix} \mu_{x1} \\ \vdots \\ \mu_{x8} \end{bmatrix} \tag{5.4}$$

$$\boldsymbol{f_1} = \begin{bmatrix} -c & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & -c \end{bmatrix} \cdot \begin{bmatrix} \mu_{x1} \\ \mu_{x2} \\ \vdots \\ \mu_{x7} \\ \mu_{x8} \\ \hline \mu'_{x1} \\ \vdots \\ \mu'_{x8} \end{bmatrix} + \begin{bmatrix} v_{1a} \\ v_{1a} \\ \vdots \\ v_{1d} \\ v_{1d} \\ \hline 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \boldsymbol{g_1} = \begin{bmatrix} \mu_{x1} \\ \vdots \\ \mu_{x8} \end{bmatrix} \tag{5.5}$$

c = Convergence parameter
μ_{x_n}, μ'_{x_n} = Belief of x position and the belief of the first derivative $\dot{x}$ of the carts
$\mu_{a\dots f}, \mu'_{a\dots f}$ = Alphabetical letters denote the beliefs of position and velocity of the higher level goals
$v_{a\dots f}$ = The causal states of the goal positions

The generative model has a representation of 8 carts, represented as positions on a 1-dimensional axis (μ_x). The model also defines 'goals' which are enumerated alphabetically (μ_{2a}, μ_{2b}...). These are positions that the carts move towards in groups of two. These goals themselves are attracted to higher goals (μ_{3e}, μ_{3f}) represented in a higher level. Those in turn move towards an end goal position (η_g). These goals have no real-world counterparts, they merely represent an internal belief of the agent about the behaviour of the carts. This internal belief of the agent will enforce the same hierarchical dynamics on the 'real' carts. Each state in this model behaves according to the same dynamics that assume they converge on a given goal (v). This means they have the same form of

dynamic function f. The dynamic behaviour of the states in the first level are described by the function $\boldsymbol{f_1}$, and the predicted sensory states in $\boldsymbol{g_1}$ 5.5. Those of the second and third levels are described in the same goal-oriented way (5.4, 5.3).

The hierarchical model described here assumes that the causal and sensory states are the same as the dynamic states, with no additional mapping between them. Therefore the role of the generative functions g is simply to select the appropriate states as the predictions for the causal states v of the level below it. At the lowest level these are predictions for the sensory states s. This is similar to the C matrix in the physical system (5.2), it simply means that the dynamic states at each level are the predictions for the causal states. The dynamic states at the lowest level are the predictions for the sensory states. This means the agent assumes the hidden states to be directly observable, which they are.

Note that the dynamics described in the generative model do not represent a system that could be commonly encountered in the real world. The equations for the accelerations describe a form of damping, but this is complemented with equations for the velocity that determine the convergence on the goal position.

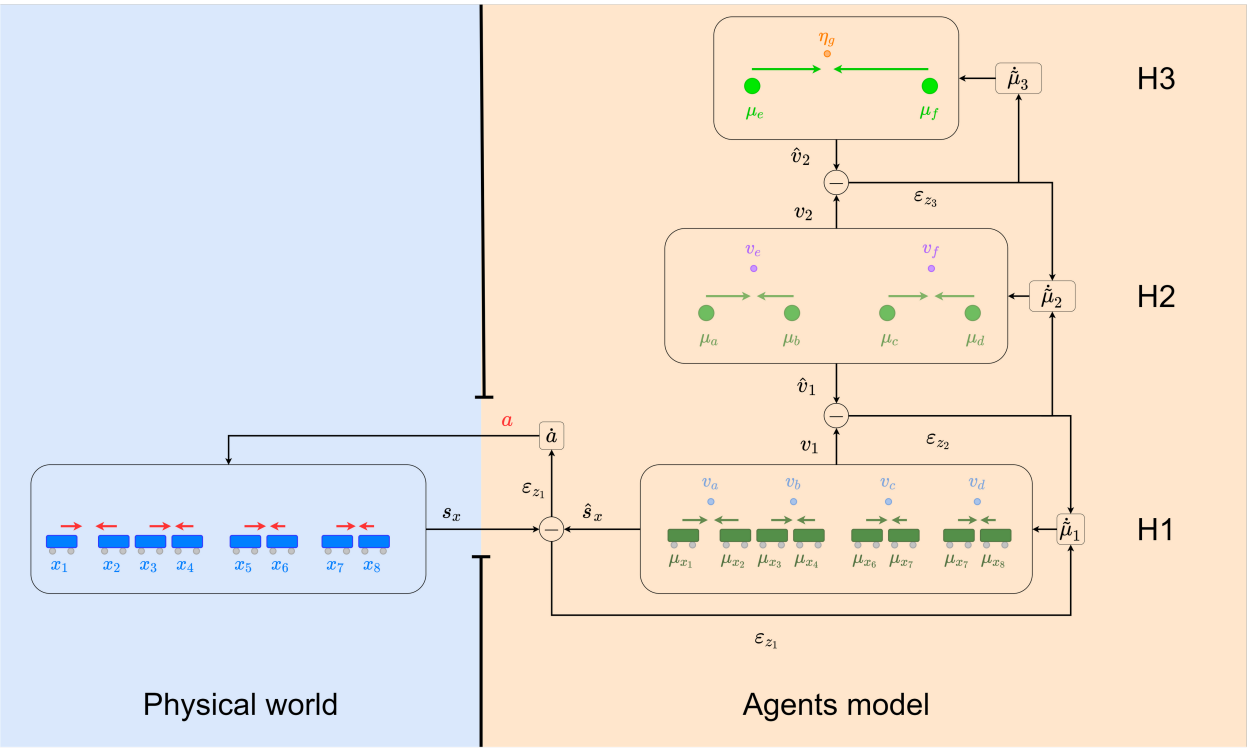


Figure 5.1: External world consisting of 8 pushable carts, and the internal model of the agent with the dynamic states μ and the causal states v. There are prediction errors for both the μ and the v, but only those for v are shown here.

$$
\begin{aligned}
\hat{\tilde{\boldsymbol{\mu}}}_{\mathbf{3}}' &= \boldsymbol{f}_3(\boldsymbol{\mu}_3, \eta) \\
\hat{\boldsymbol{v}}_{\mathbf{2}} &= \boldsymbol{g}_3(\boldsymbol{\mu}_3)
\end{aligned}
\qquad
\hat{\tilde{\boldsymbol{s}}} = \begin{bmatrix} \hat{s} \\ \hat{s'} \end{bmatrix}
\qquad
\tilde{\boldsymbol{\mu}}_{\mathbf{1}} = \begin{bmatrix} \mu_{x1} \\ \vdots \\ \mu_{x8} \\ \hline \mu'_{x1} \\ \vdots \\ \mu'_{x8} \end{bmatrix}
\tilde{\boldsymbol{\mu}}_{\mathbf{2}} = \begin{bmatrix} \mu_{2a} \\ \vdots \\ \mu_{2d} \\ \hline \mu'_{2a} \\ \vdots \\ \mu'_{2d} \end{bmatrix}
\quad
\tilde{\boldsymbol{\mu}}_{\mathbf{3}} = \begin{bmatrix} \mu_{3e} \\ \mu_{3f} \\ \hline \mu'_{3e} \\ \mu'_{3f} \end{bmatrix}
$$

$$
\begin{aligned}
\hat{\tilde{\boldsymbol{\mu}}}_{\mathbf{2}}' &= \boldsymbol{f}_2(\boldsymbol{\mu}_2, \boldsymbol{v}_2) \\
\hat{\boldsymbol{v}}_{\mathbf{1}} &= \boldsymbol{g}_2(\boldsymbol{\mu}_2)
\end{aligned}
\qquad (5.6)
$$

$$
\begin{aligned}
\hat{\tilde{\boldsymbol{\mu}}}_{\boldsymbol{x}}' &= \boldsymbol{f}_1(\boldsymbol{\mu}_x, \boldsymbol{v}_1) \\
\hat{\boldsymbol{s}} &= \boldsymbol{g}_1(\boldsymbol{\mu}_x)
\end{aligned}
\qquad
\boldsymbol{v}_{\mathbf{1}} = \begin{bmatrix} v_{1a} \\ \vdots \\ v_{1d} \end{bmatrix}
\quad
\boldsymbol{v}_{\mathbf{2}} = \begin{bmatrix} v_{2e} \\ v_{2f} \end{bmatrix}
\boldsymbol{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_8 \end{bmatrix}
\qquad (5.7)
$$

5.2.1 State Updates

Now that the generative model is defined, the corresponding update rules can be applied using the principle of gradient descent on the free energy (5.9). The Free Energy is expressed in terms of the prediction error and then the partial derivatives with respect to the states μ, v and s are take to determine the updates of the internal beliefs and the actions. This is done in the same way as is described in chapter 2.

$$
\begin{aligned}
\boldsymbol{\varepsilon_\omega} &= (D \cdot \boldsymbol{\mu}) - \boldsymbol{f} \\
\boldsymbol{\varepsilon_z} &= \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{s} \end{bmatrix} - \boldsymbol{g} \\
F &= \frac{1}{2}\boldsymbol{\varepsilon}_{\boldsymbol{\omega}}^T \cdot \boldsymbol{\Pi_\omega} \cdot \boldsymbol{\varepsilon_\omega} + \frac{1}{2}\boldsymbol{\varepsilon}_{\boldsymbol{z}}^T \cdot \boldsymbol{\Pi_z} \cdot \boldsymbol{\varepsilon_z} + \frac{1}{2} log(|\boldsymbol{\Pi_\omega} \cdot \boldsymbol{\Pi_z}|)
\end{aligned}
\qquad (5.8)
$$

$$
\begin{aligned}
\dot{\boldsymbol{\mu}} &= D \cdot \boldsymbol{\mu} - \kappa_\mu \cdot \frac{\delta F}{\delta \boldsymbol{\mu}} \\
\dot{\boldsymbol{v}} &= -\kappa_v \cdot \frac{\delta F}{\delta \boldsymbol{v}} \\
\dot{\boldsymbol{a}} &= -\kappa_a \cdot \frac{\delta \boldsymbol{s}}{\delta \boldsymbol{a}} \cdot \frac{\delta F}{\delta \boldsymbol{s}}
\end{aligned}
\qquad (5.9)
$$

$$\boldsymbol{\varepsilon_\omega} = (\boldsymbol{D}\cdot\boldsymbol{\mu}) - \boldsymbol{f} = \begin{bmatrix} \mu'_{3e} \\ \mu'_{3f} \\ \hline 0 \\ 0 \\ \hline \mu'_{2a} \\ \vdots \\ \mu'_{2d} \\ \hline 0 \\ \vdots \\ 0 \\ \hline \mu'_{x1} \\ \vdots \\ \mu'_{x8} \\ \hline 0 \\ \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} -c\cdot\mu_{3e} + c\cdot\eta_g \\ -c\cdot\mu_{3f} + c\cdot\eta_g \\ \hline -c\cdot\mu'_{3e} \\ -c\cdot\mu'_{3f} \\ \hline -c\cdot\mu_{2a} + c\cdot v_{2e} \\ \vdots \\ -c\cdot\mu_{2d} + c\cdot v_{2f} \\ \hline -c\cdot\mu'_{2a} \\ \vdots \\ -c\cdot\mu'_{2d} \\ \hline -c\cdot\mu_{x_1} + c\cdot v_{1a} \\ \vdots \\ -c\cdot\mu_{x_8} + c\cdot v_{1d} \\ \hline -c\cdot\mu'_{x_1} \\ \vdots \\ -c\cdot\mu'_{x_8} \end{bmatrix} \qquad \boldsymbol{\varepsilon_z} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{s} \end{bmatrix} - \boldsymbol{g} = \begin{bmatrix} v_{2e} \\ v_{2f} \\ \hline v_{1a} \\ \vdots \\ v_{1d} \\ \hline s_{x1} \\ \vdots \\ s_{x8} \\ s'_{x1} \\ \vdots \\ s'_{x8} \end{bmatrix} - \begin{bmatrix} \mu_{3e} \\ \mu_{3f} \\ \hline \mu_{2a} \\ \vdots \\ \mu_{2d} \\ \hline \mu_{x1} \\ \vdots \\ \mu_{x8} \\ \mu'_{x1} \\ \vdots \\ \mu'_{x8} \end{bmatrix} \tag{5.10}$$

The simulation is done without considering noise. This means that there is still no information about the noise parameters which can be used to determine the values of the precision matrices. The precision matrices are therefore taken to be diagonal, with parameters that can be chosen to tune the system.

$$\boldsymbol{\Pi_\omega} = \begin{bmatrix} \pi_{\omega_1} & 0 & \dots \\ 0 & \pi_{\omega_2} & \\ \vdots & & \ddots \end{bmatrix} \boldsymbol{\Pi_z} = \begin{bmatrix} \pi_{z_1} & 0 & \dots \\ 0 & \pi_{z_2} & \\ \vdots & & \ddots \end{bmatrix} \tag{5.11}$$

As an example, the partial derivatives of the Free energy with respect to s_1, μ_{x_1} and v_{1a} are given (5.12). This demonstrates what kind of expressions are used in updating the states of the agent's internal model.

$$\begin{aligned} \frac{\delta F}{\delta \mu_{x1}} =& \frac{1}{2}\left[\pi_{\omega_{x'_1}}(2\cdot -c(\mu'_{x1} + c\cdot\mu_{x1} - c\cdot v_{1a}))\right] + \frac{1}{2}\left[\pi_{z_{s_1}}\cdot 2\cdot -1(s_1 - \mu_{x1})\right] \\ \frac{\delta F}{\delta v_{1a}} =& \frac{1}{2}\left[\pi_{\omega_{x'_1}}\cdot 2\cdot c(\mu'_{x1} + c\cdot\mu_{x1} - c\cdot v_{1a}))\right] + \frac{1}{2}\left[\pi_{z_{v_{1a}}}\cdot 2\cdot 1(v_{1a} - \mu_{2a})\right] \\ \frac{\delta F}{\delta s_{x1}} =& \frac{1}{2}\left[\pi_{z_{s_{x1}}}\cdot 2\cdot 1(s_{x1} - \mu_{x1})\right] \\ \frac{\delta F}{\delta s'_{x1}} =& \frac{1}{2}\left[\pi_{z_{s'_{x1}}}\cdot 2\cdot 1(s'_{x1} - \mu'_{x1})\right] \end{aligned} \tag{5.12}$$

Once all the partial derivatives are derived as in (5.12), they can be used to construct the update rules as in (5.9). For this an expression for the relation between s and a is needed in the form of

an $\boldsymbol{R}$ matrix, in order to obtain appropriate update formulas for the actions. In this case a simple proportional relation is assumed between a and s, s'. this means an increase in the action a is expected to result in an increase in both s and s' by the same amount. This assumption results in a diagonal matrix of the form:

$$\boldsymbol{R} = \frac{\delta \boldsymbol{s}}{\delta \boldsymbol{a}} = \begin{bmatrix} 1 & 0 & \\ 0 & 1 & \\ & & \ddots \\ 1 & 0 & \\ 0 & 1 & \\ & & \ddots \end{bmatrix} \tag{5.13}$$

This mapping from actions to sensory states is one of the bold assumptions that is made in active inference. The assumptions about the effect of the actions are inaccurate, but they result in control actions on the sensory prediction errors. It is essentially the point where the internal dynamics of the agent switches from a forward model generating predictions to an inverse model generating actions from prediction errors. It has been proposed that this inverse mapping generating actions is realized in biological systems using reflex arcs, and that this can be used in the same way in robots ([1], [23]). However, this is an issue that needs to be studied more thoroughly to determine how this can be optimally put to use for robot control.

The update expressions are then derived using the gradient on the free energy. Filling in the state update equations 5.9 yields the resulting update expressions. 5.14 shows updates for the variables μ_{x_1}, v_{1a} and a_1 as an example. Note that the subscript a is used in v_{1a} and μ_a as an alphabetical designation, but in $\dot{a}_1$ and κ_a it refers to the actions.

$$\begin{aligned} \dot{\mu}_{x_1} &= \mu'_{x_1} - \kappa_\mu \cdot \left[\pi_{\omega_{x'_1}} (-c(\mu'_{x1} + c \cdot \mu_{x1} - c \cdot v_{1a})) + \pi_{z_{s_1}} \cdot (\mu_{x1} - s_1) \right] \\ \dot{v}_{1a} &= -\kappa_v \cdot \left[\pi_{\omega_{x'_1}} \cdot c(\mu'_{x1} + c \cdot \mu_{x1} - c \cdot v_{1a})) + \pi_{z_{v_{1a}}} \cdot (v_{1a} - \mu_{2a}) \right] \\ \dot{a}_1 &= -\kappa_a \cdot \left[\pi_{z_{s_{x1}}} \cdot (s_{x1} - \mu_{x1}) + \pi_{z_{s'_{x1}}} \cdot (s'_{x1} - \mu'_{x1}) \right] \end{aligned} \tag{5.14}$$

Together with the equations of motion for x, these update expressions form the differential equations that describe the behaviour of the whole system, consisting of the real world, the internal model of the agent and the actions that the agent exerts on the world.

5.3 Results

The generative model of the cart groups and the steps described above have been put into code and the update expressions have been integrated with a differential equation solver using a Runge-Kutta algorithm. The choice of the learning rates $\kappa_\mu, \kappa_v, \kappa_a$ and the precision terms in the matrices Π_ω, Π_z had a significant effect on how quickly and smoothly the states converged to a solution. Choosing parameters that resulted in desirable behaviour once again required some trial and error. In the

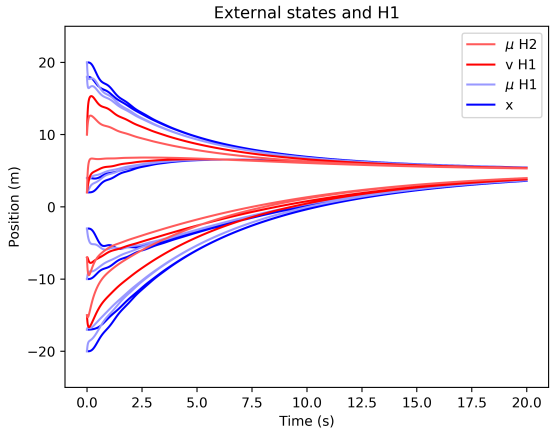
end the following parameter values were chosen for the learning rates: $\kappa_\mu = 1, \kappa_v = 1, \kappa_a = 10$. This means the actions were updated more aggressively than the causal states v and the dynamic states μ. The dynamic precision elements π_ω in Π_ω were all set to 1. The causal precision elements π_z in Π_z were all set to 10. This means that the prediction error on the causal states was weighed more heavily in the state updates. In order to lower the time in which the all the carts could reach the goal, the converging parameter c for all levels was set to 3 instead of 1. Nonetheless, the system converged very slowly and effective tuning of the parameters remains an issue for investigation.

Figure 5.2 shows the response of the states in the simulation. The results (5.2a) and (5.2b) demonstrate that the goal-seeking behaviour of the carts as dictated by the generative model works as expected, and the carts converge in ever larger groups and eventually end up at the expected location. This result is to be expected, and it is clear how higher level states can affect the behaviour of low-level states and eventually the actual carts. What is less obvious is how the outside world affects the internal model. Minimizing the free energy means adapting the internal model to the outside world as well as conforming the outside world to the model. To highlight this mechanism, the same simulation is run while the leftmost cart was disabled. The actions exerted by the agent now no longer have an effect on that cart. The results for this are shown in figures (5.2c) and (5.2d). The higher-level states are influenced both by the believed convergence towards the end goal, and by the measured location of the carts. If the internal model is not reconcilable with the actual behaviour of the world (one cart is stuck), the final value of the states will be a compromise between the two. This means that the higher level beliefs that concern the impaired cart will end up somewhere between the goal and the actual position of that cart.

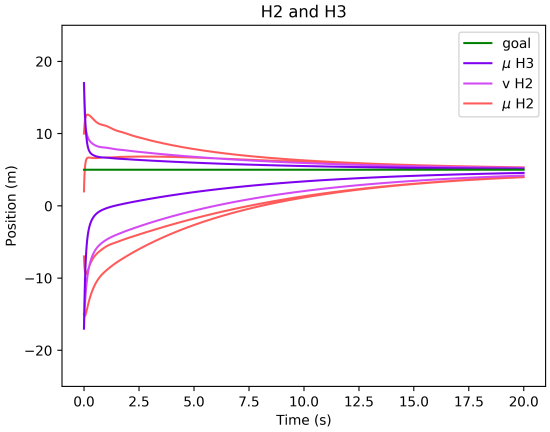
As we can see, the cart x_1 is disabled and stays still. The cart x_2 is activated but does not converge to the end goal location, because the sub-goal μ_a is also affected by the 'stuck' cart. A graphical representation of the final position of the sub goals and the carts can be seen in figure 5.3b. The effort that the agent exerts on the first cart a_1 rises to a very large value. This is because the prediction error on x_1 persists, and therefore the action update $\dot{a}$ stays the same. During integration this produces very large values.

Figure 5.2d shows the effects that disabling one cart has at higher levels in the model. We observe that the states that rely on or are related to the disabled $x1$, do not converge to the end goal value of 5 as they did before. This is because they are finding a compromising solution between the predicted goal and the actual values of the cart locations. This shows what kind of consequences can emerge from a discrepancy between the model employed by the agent and the actual physical system, and how this mismatch affects different layers of the model.

Functional Hierarchical Cart System

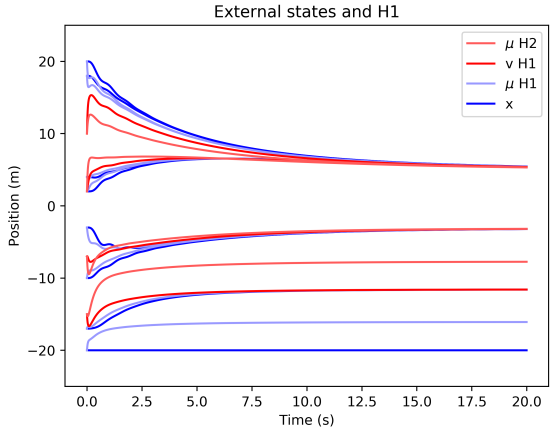


(a) External states $\boldsymbol{x}$ and the dynamic and causal beliefs $\boldsymbol{\mu}, \boldsymbol{v}$ at level $H1$ and the dynamic beliefs ,$\boldsymbol{mu_2}$, at level H_2.

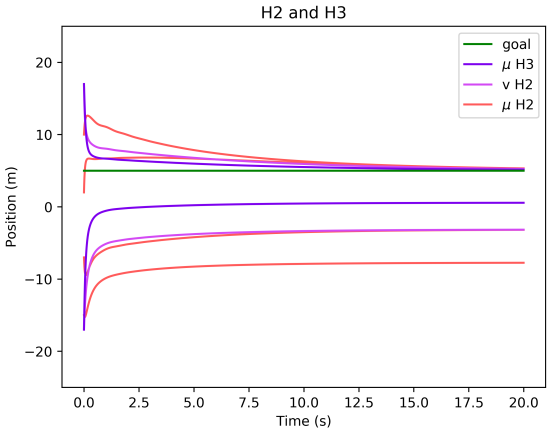


(b) The internal beliefs at higher levels. $\boldsymbol{\mu_2}, \boldsymbol{v_2}, \boldsymbol{\mu_3}$.

Impaired Hierarchical Cart System

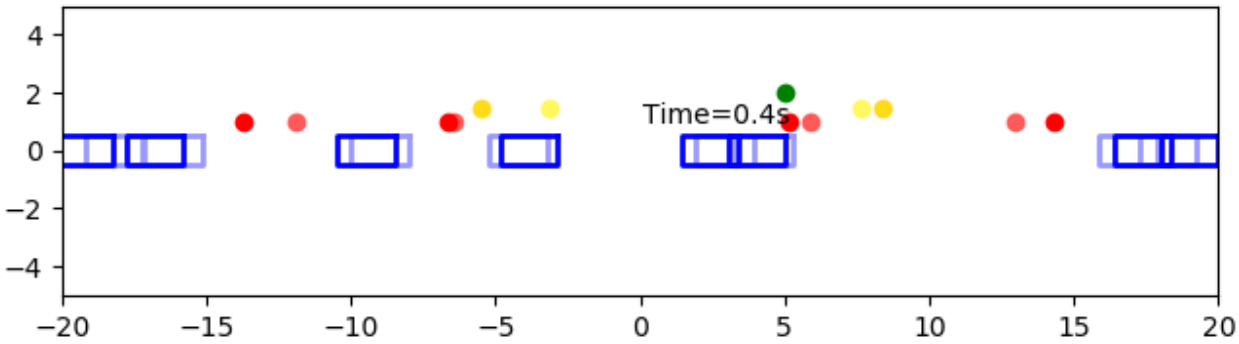


(c) External states and internal beliefs for an impaired system.

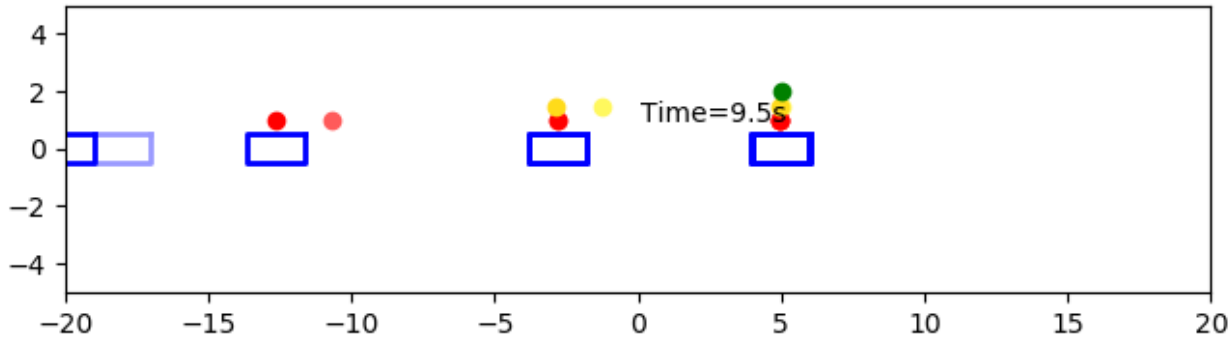


(d) Beliefs at higher levels for an impaired system.

Figure 5.2: The external and internal states over time for a functional system and an impaired system. Parameter values: $\kappa_\mu = 1, \kappa_v = 1, \kappa_a = 10$, $\pi_{\omega_{1\dots n}} = 1$, $\pi_{z_{1\dots n}} = 10$, $c = 3$. For an intuitive representation of the levels see the screenshot of the animation (5.3)



(a)



(b)

Figure 5.3: Screen shots of the animation at time $t = 0.4$ and $t = 9.5$, produced by the simulation which provide a graphical representation of the carts and the internal states. The blue squares represent the carts x. The dots represent the causal states $\boldsymbol{v}$, the red dots being $\boldsymbol{v_1}$, the yellow dots $\boldsymbol{v_2}$ and the green dot being the prior η. The lighter squares and dots represent the dynamic states $\boldsymbol{\mu_x}$, $\boldsymbol{\mu_{g1}}$ and $\boldsymbol{\mu_{g2}}$

5.4 Chapter Summary

In this chapter the active inference based algorithm has been implemented on a simple control problem using a hierarchical model. The model incorporated some hierarchical behaviour that was not present in the physical world, which in this case consisted of 8 pushable carts. The significance of this hierarchical representation is that the agent is able to divide the action and perception capabilities into different layers. This means that the agent can 'summarize' its sensory information at higher levels and obtain representations that condense a larger volume of information into a single value. Crucially, the agent can also set goals for this summarized representation and the lower levels of the internal model will automatically react to achieve the desired goal. This demonstrates how agents can 'delegate' sensory processing and action in processes at lower levels, resulting in efficient handling of large volumes of information.

The example in this chapter shows a simple but practical realization of this behaviour. The higher level end goal provided a 'summarized' task for the actions on the carts. This task was realized by dividing it into smaller tasks that involved smaller groups of carts. The implementation given in this chapter is purely for demonstration purposes. Clearly there are more optimal ways to direct 8 carts towards an end goal. The point of the cart groups model was to show how active inference using hierarchical models might work in practice.

The results illustrate that the agent is able to realize the desired behaviour in the physical system up to a point. The top-down signals in the internal model function to conform the lower level states to higher level beliefs and goals, and eventually dictate actions on the world (the carts). The bottom-up signals adapt the higher level states to the lower level states, dictated by evidence from sensory signals. These two streams of signals, both produced by the prediction error, work together to match the internal model to the physical system, and vice-versa. When there is an unavoidable discrepancy between the two, a compromising solution is found that minimizes the prediction error for the agent. This has been demonstrated by disabling one of the 8 carts and letting the simulation converge.
The cart groups example offers an insight into the practical application of the method described in previous chapters with a hierarchical model, and has shown some of the behaviour that can be expected of such an algorithm. The effective use of such an algorithm may not yet be clear, but now that the mechanism of the method has been demonstrated, it can be applied to more practical applications.

Chapter 6

Robot Arm

So far the active-inference based algorithm has been demonstrated on simple 1-dimensional control problems with a single level generative model (chapter 4), and a hierarchical generative model (chapter 5). Both of these cases implement a form of goal-reaching behaviour that is determined by the dynamics expressed in the generative models. The hierarchical case implemented different levels of goals, where the number of goals decreased in higher levels. The question remains how this hierarchical description could be relevant in practical applications. Can a hierarchical model be used in a practical scenario to enforce the proper reactions in a control problem? If so, what role does the hierarchical aspect of the generative model fulfil and how is its function important to the implementation of the algorithm? To investigate these points, the control algorithm will be applied to a control problem that, albeit simple, is more representative of real-life robot control problems. To this end, this chapter will consider a reaching task in which a 2-D robot arm with two segments has to position its end effector. The goal of this chapter is:

> Subgoal 3: Apply the algorithm to a 2-D robot arm with various hierarchical generative models to investigate the function of the hierarchical model for active inference in robot control.

There are two reasons the robot arm was chosen to test the algorithm. The first is that a (robot) arm has been used before in literature as an example for motor control using active inference [22], [14]. These papers did not focus on hierarchy in the models, and so they provide suitable references for the implementations of hierarchical models that is done here.

The second reason for simulating a 2-D robot arm is that it resembles problems that can be encountered in real-world robotics. It is a non-linear dynamic system where the effects of actions can be quite complex. There is a clear goal for which there are multiple valid configurations and there is a non-linear relation between this goal (position) and the observations and actuation (angles). These are all characteristics that are relevant for real-world robot control systems.

6.1 Chapter Overview

This chapter considers a 2-D robot arm as an example of a practical robotic application, and a hierarchy is defined that includes high-level representations that can be deduced from sensory states. An exploratory investigation is carried out to find a suitable hierarchical model that enables the active inference algorithm to impose the desired behaviour on the robot arm. The robot arm consists of two segments with actuated joints. The sensory information that is available to the agent are the joint angles and angular velocities. The agent can exert actions (torques) on the joints. The goal of the agent is to get the end effector (the end point of the second segment of the arm) to a specific goal location $[x_g, y_g]$.

The goal of this simulation is not to demonstrate a more effective control algorithm for robotic arms, high-performance control solutions already exist for this kind of problem. Rather, the aim of this chapter is to analyse the performance of this method and to show that active inference using a hierarchical model can be implemented on an arbitrary but relevant robotic mechanism.

For the simulations of the robot arm in this chapter, the hierarchical distinction is made between the sensory information of the joint angles ϕ and the second-level representations of the positions of the elbow joint and end effector in x, y coordinates.

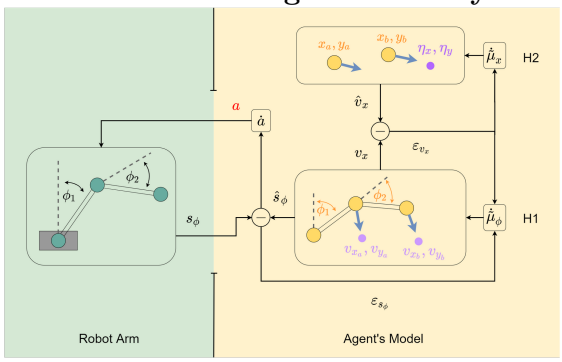
The goal-reaching behaviour of the robot arm is determined by the internal model. In this study, we have analysed five ways to construct this model, see figure 6.1. The first three examples include representations for joint and end effector coordinates at the second level ($\mu_{x_a}, \mu_{y_a}, \mu_{x_b}, \mu_{y_b}$), and the joint angles at the first level (ϕ_1, ϕ_2). This means that there are more variables in the second level (4 variables) than in the first (2). This seems opposed to the purpose of hierarchy as has been implemented in the group cart example in the previous chapter, where the hierarchy serves to condense information in higher levels. Indeed, as we will discuss in section 6.6, this increase of variables at higher levels can cause some issues.

Although counter-intuitive, this approach has been pursued to explore ways that hierarchy might be used to process sensory information and implement control. Given only the joint angles, it seems logical that the end positions of the links could be valuable information to an agent. Therefore it is worth exploring how these might be inferred from lower level representations in a hierarchical model. The first three examples in this chapter explore if and how such an approach can be effective and we discusses the issues that emerge from the simulations.

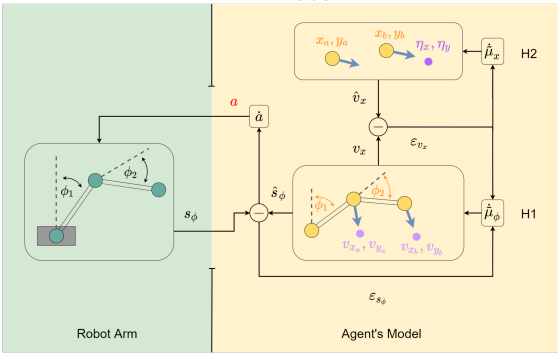
The images in figure 6.1 show the different hierarchical models that have been investigated. The first two cases consider a hierarchy in which the second hierarchical level, which is referred to as $H2$, contains representations of the positions of the arm and the goal-directed dynamics. The aforementioned issues that these two experiments raise are addressed in the next two cases, which take into consideration the degrees of freedom and necessary constraints in the second hierarchical level. Finally, after examining the relevance of the hierarchical model for this particular control problem, a different approach is taken in which the hierarchy is used to add an extra layer of complexity to the task of the robot. Readers interested in a successful hierarchical implementation with added complexity in the robot arm are urged skip to section 6.5. Those interested in the likely obstacles in similar implementations are advised to read the full account of the exploratory attempts in section 6.3 and 6.4

Introducing Hierarchy

1. Virtual Angular Velocity

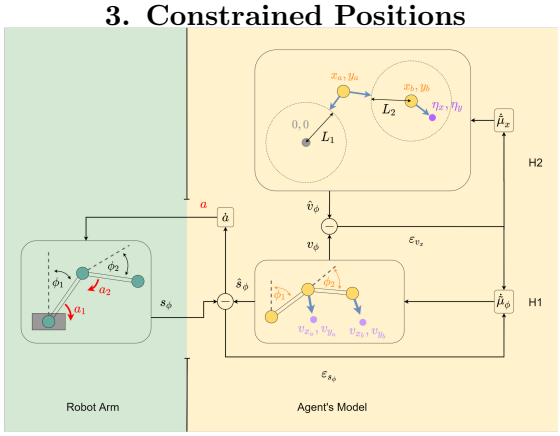


2. Arctan

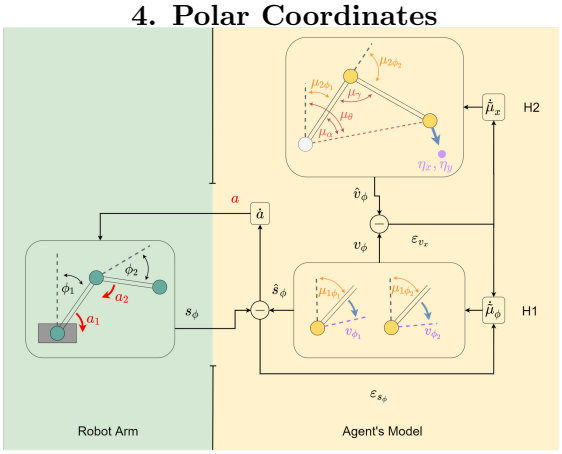


Considering Constraints

3. Constrained Positions



4. Polar Coordinates



Hierarchical Task Expansion

5. Vector Field Path

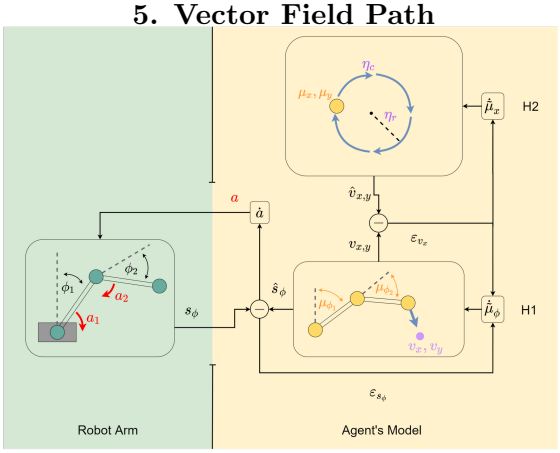


Figure 6.1: The five hierarchical models that are discussed in each section in this chapter.

6.2 Physical model

The physical model (generative process) of the robot arm is described by a set of non-linear differential equations, that are significantly more complicated than the 1-dimensional linear cart equations. As was mentioned in section 3.2.1, these equations are determined in a separate piece of code that implements the TMT method ([25]). This method involves a transformation to independent coordinates (ϕ_1, ϕ_2) to calculate an efficient expression for the angular accelerations. This determines the expressions for M and G in the general differential equation $M\ddot{\boldsymbol{\phi}} = G$ that describes most physical systems. M is a reduced mass matrix and G is a reduced force vector. With these matrices the differential equation could be solved for $\ddot{\phi}$. The equations were found to be correct by checking if the simulation developed as a double pendulum for some simple situations.

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ \ddot{\phi}_1 \\ \ddot{\phi}_2 \end{bmatrix} = \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \\ ddx_1(\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2, a_1, a_2) \\ ddx_2(\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2, a_1, a_2) \end{bmatrix} \quad (6.1) \qquad \boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ \dot{s}_1 \\ \dot{s}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \phi_1 \\ \phi_2 \\ \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix} \quad (6.2)$$

$\phi_1, \phi_2, \dot{\phi}_1, \dot{\phi}_2, \ddot{\phi}_1, \ddot{\phi}_2$: Angular position, velocity and acceleration
ddx_1, ddx_2 : Functions describing the angular acceleration of the joints.
$s_1, s_2, \dot{s}_1, \dot{s}_2$: sensory observations of angular position and velocity
a_1, a_2 : action, in this case joint torque

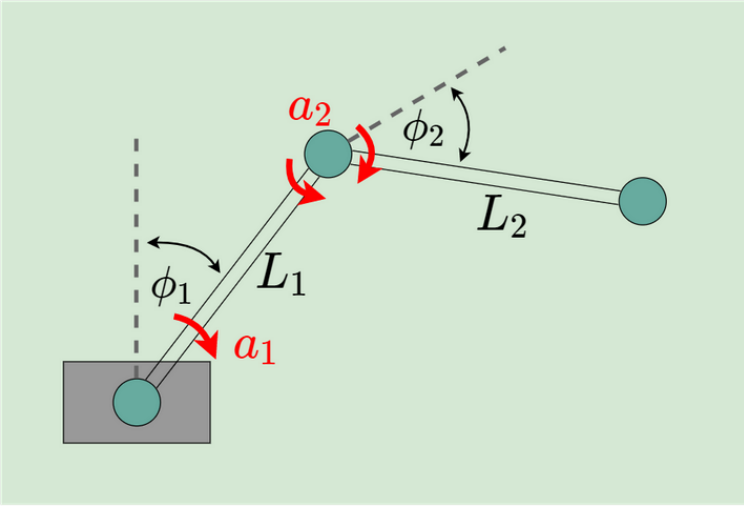


Figure 6.2: The simulated robot arm system. The joint angles ϕ_1 and ϕ_2 (and their velocities) are available to the agent as sensory states. The two joints are actuated by action torques a_1 and a_2 provided by the agent.

Note that for most investigations into active inference state noise terms are included in equation 6.1 and sensory noise terms for equation 6.2. Here these noise terms are excluded for clarity as this thesis focuses on the investigation into hierarchical implementation. Gravity forces were omitted, but a damping torque on the joints was included. This made it easier to analyse the behaviour of the robot arm. The full expressions for the angular accelerations in these equations are quite extensive and have not been written out explicitly. Each of the arms had a length of $L = 1(m)$, a

mass of $M = 1(kg)$, and a moment of inertia of $I = 0.2(kg \cdot m^2)$. The joints experienced a damping coefficient of $c = 1(\frac{N \cdot m \cdot s}{rad})$.

6.3 Introducing Hierarchy

The first option that is explored is to include representations of the cartesian coordinates of the elbow joint and the end effector in the second level ($H2$). These two positions are the most obvious deductions that can be made from the sensory information of the angles, so it makes sense to represent these 'abstractions' in a second hierarchical level. 'Abstract' representations here are any representations that do not directly reflect the sensory information but have to be deduced from it. Having representations for these positions would be useful to a robot arm, as it is needed for goal reaching and obstacle avoiding behaviour.

6.3.1 Virtual Angular Velocity

In this first implementation, the second hierarchical level $H2$ contains beliefs about the x and y positions of the elbow and end effector positions of the robot arm. The dynamics described in the second level assume that the velocities of these points are proportional to the distance between the end effector and a predefined goal. This reflects the assumption that any movement made by the elbow joint will also move the end effector. These dynamics are of course not accurate to the real world dynamics, but this approach relies on the prediction error with the first level representations to provide accurate feedback. In turn, the first level relies on feedback from the sensory information coming from the joint angles.

As illustrated in figure 6.3 the first hierarchical level $H1$ relates the second level positions to the joint angles. $H1$ contains variables that represent the agents internal belief of the joint angles, μ_{ϕ_1} and μ_{ϕ_2}. The elbow and end effector positions are determined in $H2$ enter $H1$ as causal states v. The dynamics described in $H1$ move the angles μ so that these positions are realized. This is done by calculating the joint positions based on the current angles, taking the difference between this and the causal positions v, and using this to determine appropriate angular velocities. Specifically, the difference in x position is multiplied by the y position and the difference in y position is multiplied by the x position, ensuring that the angular velocity will always minimize the difference in position. This behaviour is dictated by the dynamic formula in the first level (6.4).

As in the single cart and multiple cart implementations in previous chapters, the models described in this chapter include the zero and first order states. This allows a proper comparison with the incoming first-order sensory observations $\dot{s}_1$ and $\dot{s}_2$ received by the robot arm simulation. The second order dynamic functions f' that predict the derivatives of the first-order variables are also included in the models, but these are excluded here for brevity. As in the single cart and multiple cart systems of the previous chapters, we were free to choose the values of the precision matrices and the learning rates, which was done by trial and error [1].

6.3.2 Virtual Angular Velocity Results

The simulation was conducted for two different goal positions, $[x = -2, y = 0]$ and $[x = 1, y = 1]$. The algorithm was run long enough for the system to converge to a solution, meaning that the arm assumed a static end position. The end positions can be seen in figures 6.5 and 6.8. These figures show that in both cases the algorithm has been unable to position the end effector at the desired location. The belief about the end effector position (μ_{x_b}, μ_{y_b}) in $H2$ of the internal model reaches the goal location, and the corresponding causal beliefs v_{x_a}, v_{x_b} also converge to this point. However, these values are not consistent with the beliefs for the joint angles μ_{ϕ_1}, μ_{ϕ_2}. Apparently the state updates cannot rectify this discrepancy between the beliefs of the joint angles and the beliefs of the joint locations. It becomes clear why this is when we notice that the joint angles are such that the robot arm segments are in line with the believed joint positions. Evidently this version of the generative model results in state updates that disregard the length of the segments, and only require the positions to be consistent with the direction of the segments.

The exact mechanism that leads to this stalling becomes clearer when we take a look at the update steps for the causal states v as was described in 2.41 in chapter 2. In this case, the causal states v describe the positions of the elbow joint and the end effector. $\dot{v}$ is calculated from the prediction errors. It is clear that the v terms only appear in f_1 (6.4) and g_2 (6.7), and so only the prediction errors $\varepsilon_{\omega 1}$ and ε_{z2}. Updating the v terms to minimize these prediction errors made by these functions means that the causal states will be updated to correspond with the second level positions and the first level angles. The issue here is that the prediction error $\varepsilon_{\omega 1}$ (6.9) may become zero when the causal beliefs v line up with the beliefs about the angles μ_ϕ but are not necessarily at the right distance to correspond with the real world elbow and end effector positions.

$$\dot{v} = -\kappa_v \cdot \left[\frac{\delta \boldsymbol{\varepsilon}_\omega}{\delta v} \cdot \Pi_\omega \cdot \boldsymbol{\varepsilon}_\omega + \frac{\delta \boldsymbol{\varepsilon}_z}{\delta v} \cdot \Pi_z \cdot \boldsymbol{\varepsilon}_z \right] \tag{6.8}$$

$$\varepsilon_{\omega 1} = \begin{bmatrix} \mu_{\phi_1'} \\ \mu_{\phi_2'} \end{bmatrix} - \begin{bmatrix} (v_{x_a} - x_a) \cdot y_a - (v_{y_a} - y_a) \cdot x_a \\ (v_{x_b} - x_b) \cdot (y_b - y_a) - (v_{y_b} - y_b) \cdot (x_b - x_a) \end{bmatrix} \tag{6.9}$$

This result illustrates that the feedback through prediction errors is not always effective and that a proper solution depends heavily on the type of dynamics that are included in the internal model of the agent. In this case the dynamic function in $H1$ was set up in such a way that the prediction error did not take into account the lengths of the robot arm segments.

[1] A note to those interested in the tuning of the parameters. For both the levels $H1$ and $H2$, the elements of the dynamic precision matrix Π_ω all had a value of 1 and the elements of the sensory precision matrix Π_z all had a value of 5. For these values the robot arm was found to converge relatively smoothly. The values of the learning rates κ_μ, κ_v and κ_a were set at 1, 1 and 10 respectively. The values for κ_a was higher than the other learning rates to ensure that the agent would quickly react to the prediction error from the sensory states. The dynamic precisions π_ω were lower than the sensory precisions because it was found that the robot arm would become unstable otherwise. This presumably has to do with the fact that the internal model assumes (first order) dynamics that are different from the real robot arm dynamics (second order) and do not take into account inertia and forces for example. If the agent is too confident in the internal dynamics this can cause the robot arm to lag behind the internal beliefs and may cause an unstable feedback loop. It is difficult to say if this is the real reason for the unstable behaviour, but it might explain the need for relatively high sensory precisions to keep the internal beliefs true to the real world states.

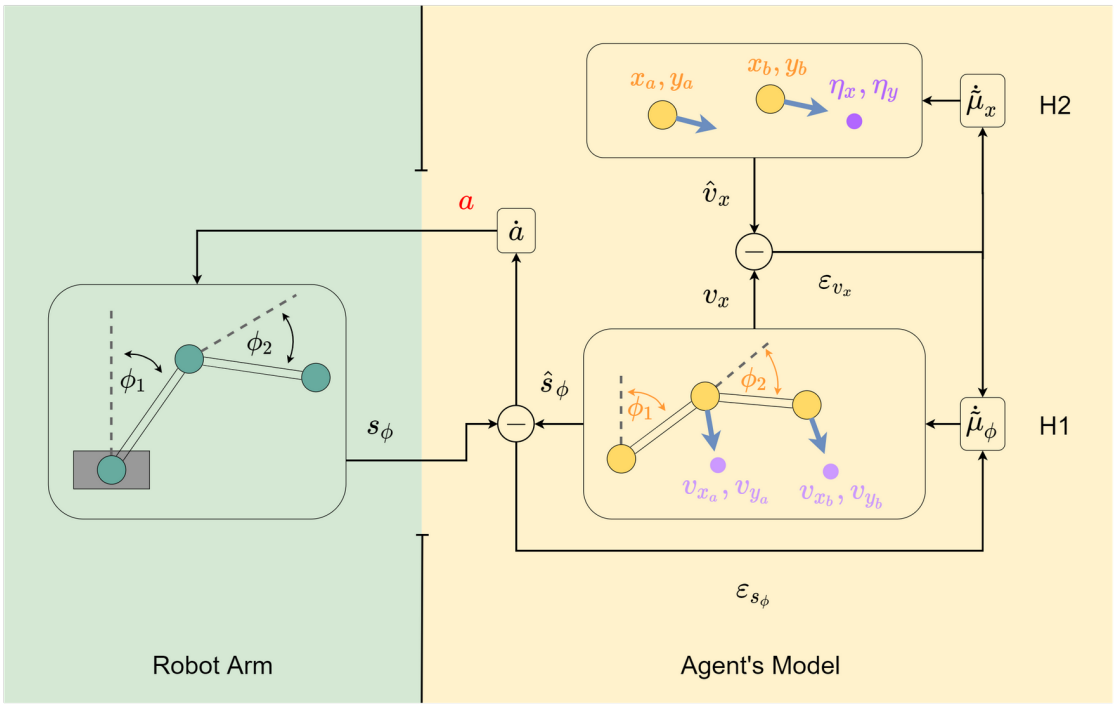


Figure 6.3: The robot arm and the internal model of the agent expressing the joint velocities $[\dot{\phi_1}, \dot{\phi_1}]$ at $H1$ based on the causal states $v_{x,y}$ determined at $H2$.

H1

$$
\begin{aligned}
x_a &= sin(\mu_{\phi_1}) \cdot L_1 \\
y_a &= cos(\mu_{\phi_1}) \cdot L_1 \\
x_b &= sin(\mu_{\phi_1}) \cdot L_1 + sin(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2 \\
y_b &= cos(\mu_{\phi_1}) \cdot L_1 + cos(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2
\end{aligned} \tag{6.3}
$$

$$
f_1 = \begin{bmatrix} \hat{\mu}_{\phi'_1} \\ \hat{\mu}_{\phi'_2} \end{bmatrix} = \begin{bmatrix} (v_{x_a} - x_a) \cdot y_a - (v_{y_a} - y_a) \cdot x_a \\ (v_{x_b} - x_b) \cdot (y_b - y_a) - (v_{y_b} - y_b) \cdot (x_b - x_a) \end{bmatrix} \tag{6.4}
$$

$$
g_1 = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}'_1 \\ \hat{s}'_2 \end{bmatrix} = \begin{bmatrix} \mu_{\phi_1} \\ \mu_{\phi_2} \\ \mu_{\phi'_1} \\ \mu_{\phi'_2} \end{bmatrix} \tag{6.5}
$$

H2

$$
f_2 = \begin{bmatrix} \hat{\mu}_{x'_a} \\ \hat{\mu}_{y'_a} \\ \hat{\mu}_{x'_b} \\ \hat{\mu}_{y'_b} \end{bmatrix} = \begin{bmatrix} \eta_x - \mu_{x_b} \\ \eta_y - \mu_{y_b} \\ \eta_x - \mu_{x_b} \\ \eta_y - \mu_{y_b} \end{bmatrix} \tag{6.6}
$$

$$
g_2 = \begin{bmatrix} \hat{v}_{x_a} \\ \hat{v}_{y_a} \\ \hat{v}_{x_b} \\ \hat{v}_{y_b} \end{bmatrix} = \begin{bmatrix} \mu_{x_a} \\ \mu_{y_a} \\ \mu_{x_b} \\ \mu_{y_b} \end{bmatrix} \tag{6.7}
$$

Figure 6.4: The equations producing the predictions by the agent with model 1, based on angular velocities. The functions f produce predictions for the dynamic states $\hat{\boldsymbol{\mu}}$. At level $H1$, function g_1 produces predictions for the sensory states $\hat{\boldsymbol{s}}$. At level $H2$, function g_2 produces predictions for the causal states $\hat{\boldsymbol{v}}$.

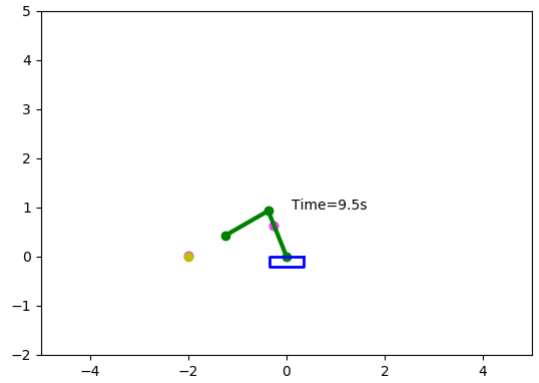


Figure 6.5: The robot arm using the virtual angular velocity model has failed to reach the goal at $[x = -2, y = 0]$.

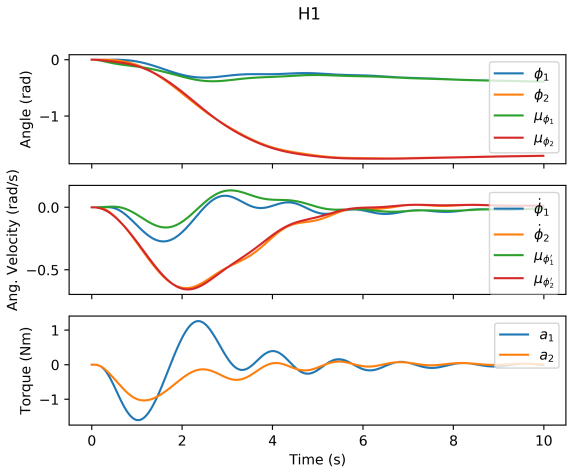


Figure 6.6: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_{\phi}, \mu_{\phi'}]$ at $H1$ for goal $[x] = -2, y = 0]$. $[a_1, a_2]$ are the action torques. The beliefs for the angles $[\mu_{\phi_1}, \mu_{\phi_2}]$ (green and red lines in the top graph) do not converge to the values $[-\pi, 0]$ that would reach the goal, so model 1 is not a successful hierarchical set-up.

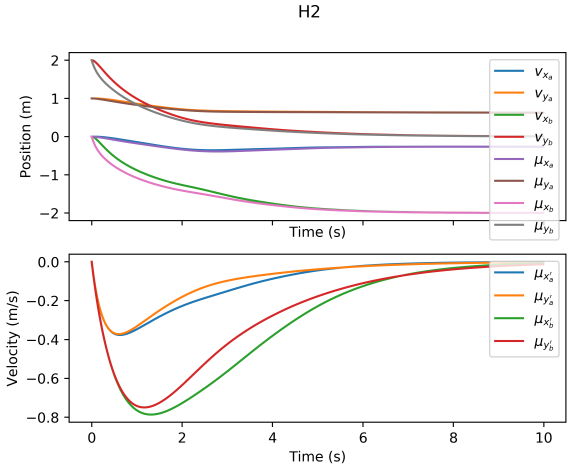


Figure 6.7: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2. The beliefs $[\mu_{x_2}, \mu_{y_2})]$ do converge to the goal, but this is not translated properly to angles at level $H1$.

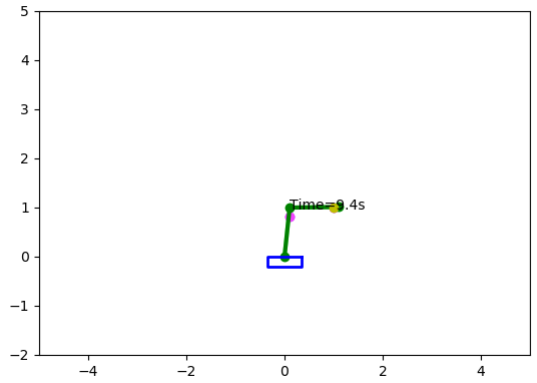


Figure 6.8: The robot arm using the virtual angular velocity model is close, but has failed to completely converge to the goal at $[x = 1, y = 1]$.

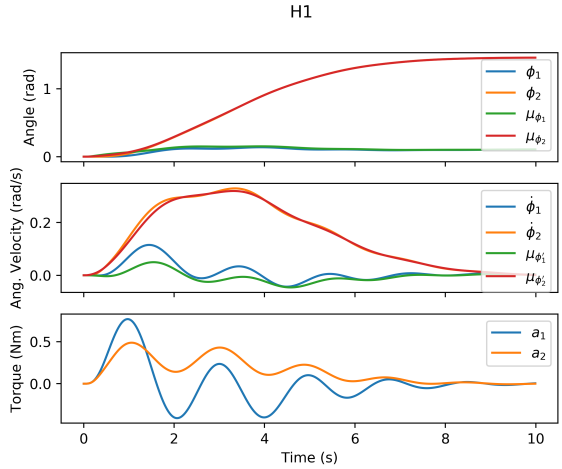


Figure 6.9: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_{\phi}, \mu_{\phi'}]$ at $H1$ for goal $[x = 1, y = 1]$. $[a_1, a_2]$ are the action torques. The beliefs for the angles $[\mu_{\phi_1}, \mu_{\phi_2}]$ (green and red lines in the top graph) do not converge to the values $[-\pi, 0]$ that would reach the goal, so model 1 is not a successful hierarchical set-up.

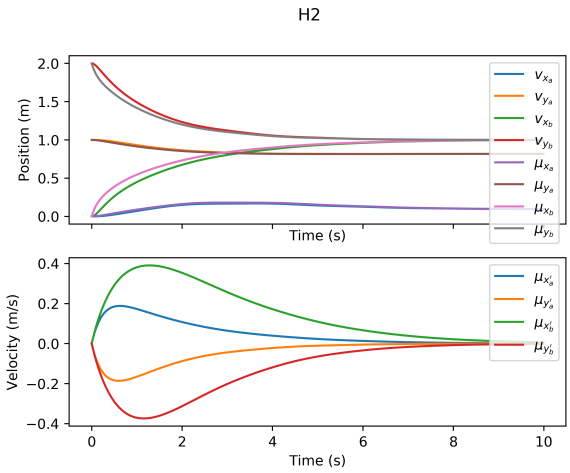


Figure 6.10: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2. The beliefs $[\mu_{x_2}, \mu_{y_2})]$ do converge to the goal, but this is not translated properly to angles at level $H1$.

6.3.3 Angles by inverse tangent

An alternative model for the agents internal model was explored, in which the causal states v represent angles and not positions. The dynamics of the positions are implemented in the second level ($H2$) in the same way as in section 6.3.1, moving towards a previously defined goal. However, the angles are are not derived in $H1$ but in $H2$ (6.13) by taking the inverse tangents ($\tan^{-1}$) of the positions of joint positions (μ_x, μ_y). These causal angles v_ϕ are used in the first level $H1$ as goals for the internal beliefs of the angles μ_ϕ, which in turn predict the angles given by the sensory signals. This model is depicted in Figure 6.11. It would also be possible to implement this type of model in a single level. After all, the angles given by g_2 could be used to directly predict the sensory signals. The objective of this chapter however, is to observe how a hierarchical model could be used to exert control on the robot arm and to investigate the behaviour that arises in this implementation. For this reason the model is implemented in two levels.

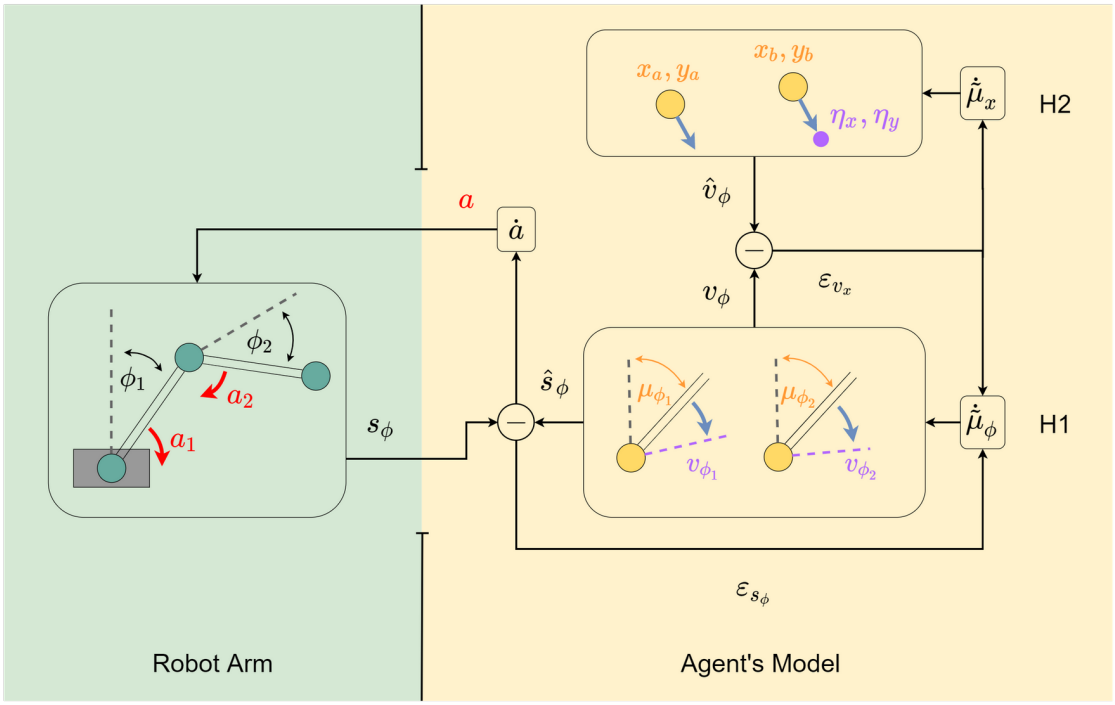


Figure 6.11: The robot arm system with the inverse tangent model. The causal states v_ϕ are predicted in the second level based on $\mu_{(x,y)}$ and represent goals for the angles.

H1

$$f_1 = \begin{bmatrix} \hat{\mu}_{\phi_1'} \\ \hat{\mu}_{\phi 2'} \\ -\mu_{\phi_1''} \\ -\mu_{\phi_2''} \end{bmatrix} = \begin{bmatrix} (v_{\phi_1} - \mu_{\phi_1}) \\ (v_{\phi_2} - v_{\phi_1}) - \mu_{\phi_2}) \\ -\mu_{\phi_1'} \\ -\mu_{\phi_2'} \end{bmatrix} \tag{6.10}$$

$$g_1 = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}_1' \\ \hat{s}_2' \end{bmatrix} = \begin{bmatrix} \mu_{\phi_1} \\ \mu_{\phi_2} \\ \mu_{\phi_1'} \\ \mu_{\phi_2'} \end{bmatrix} \tag{6.11}$$

H2

$$f_2 = \begin{bmatrix} \hat{\mu}_{x_a'} \\ \hat{\mu}_{y_a'} \\ \hat{\mu}_{x_b'} \\ \hat{\mu}_{y_b'} \end{bmatrix} = \begin{bmatrix} \eta_x - \mu_{x_b} \\ \eta_y - \mu_{y_b} \\ \eta_x - \mu_{x_b} \\ \eta_y - \mu_{y_b} \end{bmatrix} \tag{6.12}$$

$$g_2 = \begin{bmatrix} \hat{v}_{\phi_1} \\ \hat{v}_{\phi_2} \end{bmatrix} = \begin{bmatrix} tan^{-1}(\frac{\mu_{x_a}}{\mu_{y_a}}) \\ tan^{-1}(\frac{\mu_{x_b} - \mu_{x_a}}{\mu_{y_b} - \mu_{y_a}}) \end{bmatrix} \tag{6.13}$$

Figure 6.12: The equations producing the predictions by the agent with model 2, based on the inverse tangent. The functions f produce predictions for the dynamic states $\hat{\boldsymbol{\mu}}$. At level $H1$, function g_1 produces predictions for the sensory states $\hat{\boldsymbol{s}}$. At level $H2$, function g_2 produces predictions for the causal states $\hat{\boldsymbol{v}}$. The part of the function f_2 that produces estimates of the second derivatives of $\boldsymbol{\mu}$ has been excluded here for brevity.

6.3.4 Inverse Tangent Results

As before, the simulation is run for the goal positions $[x = -2, y = 0]$ and $[x = 1, y = 1]$, and some tuning was done through trial and error [2]. It quickly became apparent that this approach would only be feasible for positive x values for the beliefs of the positions. This is because the tan^{-1} function is only capable of producing values between $-\frac{1}{2}\pi$ and $\frac{1}{2}\pi$. Therefore, as soon as μ_{y_a} or $(\mu_{y_b} - \mu_{y_a})$ become negative, the corresponding predictions for the angles v_{ϕ_1} and v_{ϕ_2} flip by $\frac{1}{2}\pi$. This can be seen in figure 6.15 where v_{ϕ_1} and μ_{y_1} suddenly change direction. Figure 6.13 shows that this steers the robot arm away from the goal. This problem does not occur when the goal is placed at $[1, 1]$. However, the same issue that prevented the robot arm from reaching the goal in section 6.3.1 persists in this implementation. Again the arm segments are aligned with the corresponding goals, but do not take the segment lengths in to account. Similarly, this is because the prediction error on the angles becomes zero for when they align with the position beliefs in $H2$.

From the results in model 1, 2 and some additional unreported trials, it has become clear that it is hard, if not impossible, to implement an effective goal-reaching algorithm for the robot arm with 4 independent variables in the second level. There does not seem to be a way to represent 4 variables for joint positions in $H2$ and to match 2 variables for joint angles in $H1$. The reason it is so hard to define functions that describe the dynamic relationships between the two levels is that we are introducing extra degrees of freedom in the second level. There are multiple configurations for the joint positions $\mu_{x_a}, \mu_{y_a}, \mu_{x_b}$ and μ_{y_b} (4 degrees of freedom) that would satisfy the joint angles μ_{ϕ_1} and μ_{ϕ_2} (2 degrees of freedom) if the proper constraints are not present. The constraints in this case are the lengths of the robot arm segments that dictate the possible configurations of $\mu_{x_a}, \mu_{y_a}, \mu_{x_b}$ and μ_{y_b}. Explicitly including these constraints might therefore provide a suitable solution.

[2] The same values for the precision matrices as in section 6.3.1 were tried, namely 1 for the dynamic precisions Π_ω and 5 for the sensory (and causal) precisions Π_z. This worked when the goal position was $[1, 1]$, but for a goal at $[-2, 0]$ the simulation became very slow and got stuck during integration. Through trial and error it was found that decreasing the values of the causal precision matrix Π_{z2} to 0.5 made the simulation workable again. It is unclear what causes this difficulty in calculating the solution, but for certain situations the simulation employs debilitatingly small time steps. Adjusting the precisions did not seem to have a significant effect on the behaviour of the system in this case except for slowing down the convergence of the causal angles v_ϕ to the second level beliefs.

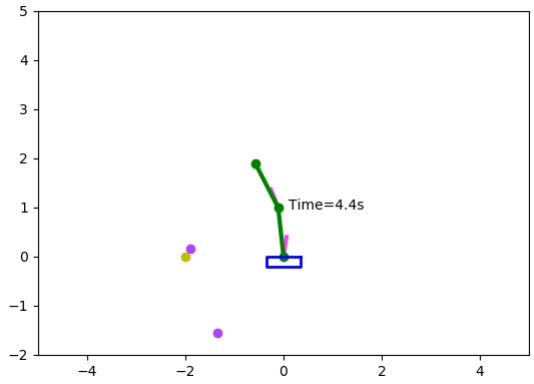


Figure 6.13: The robot arm using the inverse tangent model has failed to reach the goal at $[x = -2, y = 0]$. μ_{y_a} has gone below zero, and the model's tan^{-1} expression predicts the wrong angles. The purple dots represent second level beliefs about joint positions, the pink bars are the first level beliefs about the angles.

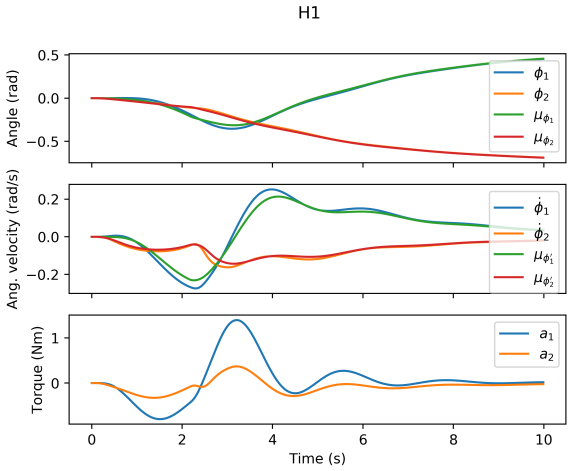


Figure 6.14: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$. $[a_1, a_2]$ are the action torques. Initially ϕ_1 moves towards the appropriate angle for the goal, but around $t = 2.2$ moves away again.

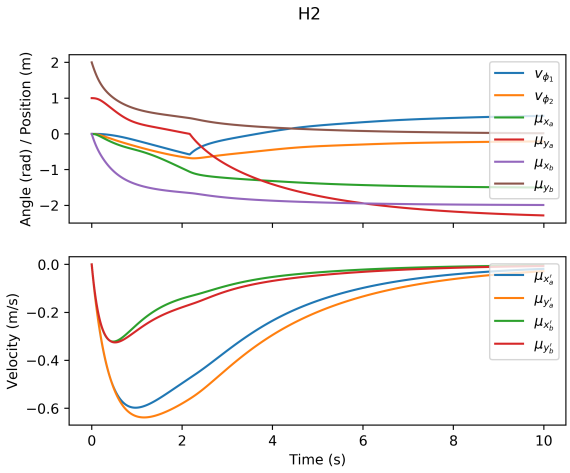


Figure 6.15: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2. Around $t = 2.2$, μ_{y_a} (red line) becomes negative and results in incorrect prediction of v_{ϕ_1}.

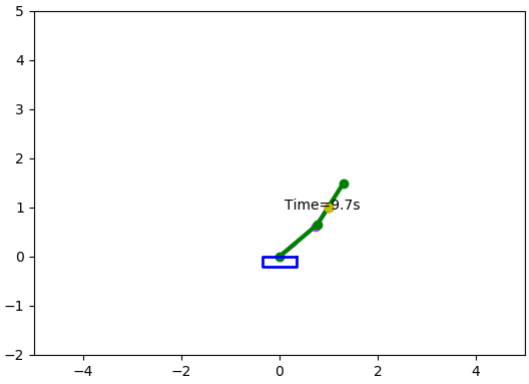


Figure 6.16: Here there is no issue with the tan^{-1} expressions but the simulation has still failed to converge to the goal at $[x = 1, y = 1]$.

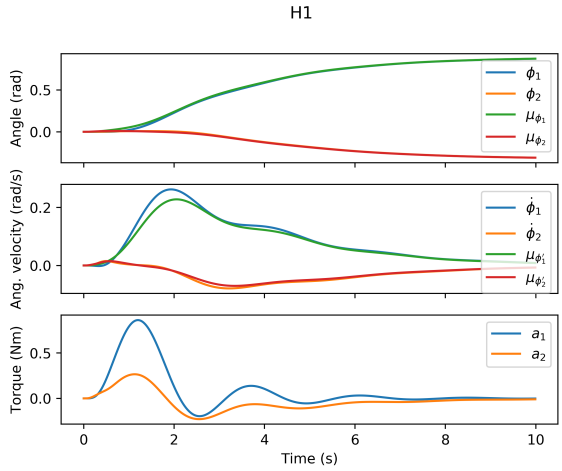


Figure 6.17: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$, these do not converge to the appropriate angels to rach the goal. $[a_1, a_2]$ are the action torques.

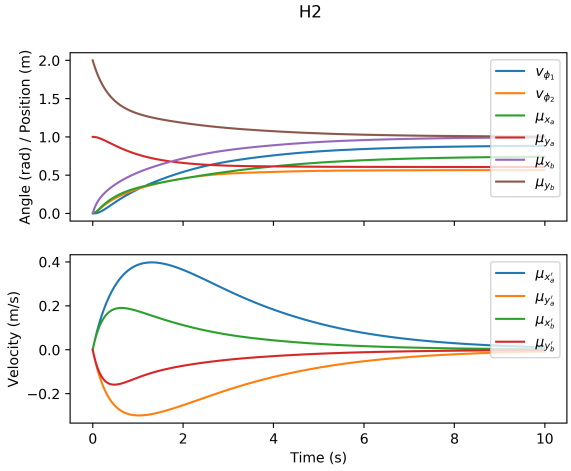


Figure 6.18: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2. The end effector belief $[\mu_{x_b}, \mu_{y_b}]$ (brown and purple line) reach the goal, but the elbow joint belief $[\mu_{x_a}, \mu_{y_a}]$ (green and red line) assumes a configuration which is not possible in the real robot arm.

6.4 Considering Constraints

The previous implementations in section 6.3 determined that higher degrees of freedom in the variables of higher levels can cause problems in predicting the corresponding lower level states and can result in the robot arm converging to an undesirable solution. A proposed solution is to introduce the necessary constraints in the model dynamics to confine the states to the appropriate configurations. In the case of the robot arm that means taking the lengths of the segments into consideration. This section explores two implementations that handle this issue differently. The first solution determines the segment angles based on only one desired position, and the second approach introduces the constraints as attracting points to the model dynamics.

6.4.1 Polar Coordinates

One way to limit the degrees of freedom in $H2$ is to simply to include less variables, by disregarding the position of the elbow joint and only representing the position of the end effector in polar coordinates in an angle μ_θ and a distance from the origin μ_r. As there are only two segments, and their length is known, the corresponding angles of the robot arm can be calculated by the so-called 'law of cosines', which states that the sides of a triangle a, b, c and an inside angle γ, are defined by the following relationship: $c = \sqrt{a^2 + b^2 - 2ab\cos(\gamma)}$ where γ is the angle opposite the side c. This formula can be inverted to produce expressions for the two segment angels as in g_2 (6.17). As in the previous implementation using the inverse tangent in section 6.3.3, the causal states v_ϕ represent attraction points for the beliefs of the angles μ_ϕ at $H1$ to move towards. Note that for most positions of the end effector, there are two configurations of the arm segments which can be employed to achieve it. One of the solutions will be with the arm 'bent' towards the left and the other with the arm 'bent' towards the right. For the internal model as it is implemented here, the robot arm will always be moving towards the 'right bending' configuration, because of assumptions that are made in using the law of cosines.

6.4.2 Polar Coordinates Results

After tuning [3]and running some trials of the simulation, the results showed that this version of the internal model was able to derive the appropriate angles to achieve the desired goal positions, as in the cases shown in figures 6.21 and 6.24. However, there were some algebraical issues with the simulation and not all situations could be evaluated by the algorithm. The simulation raised errors when the second level belief of the end effectors distance from the origin became equal or larger than the sum of the segment lengths. It is clear that the robot arm is only able to reach positions as far as when the arm is fully outstretched. This is reflected in the causal predictions g_2 (6.17). The expression $cos^{-1}(x)$ produces no valid results for $x > 1$, which occurs when $\mu_r \geq L_1 + L_2$. Because of this the initial value of μ_r could not be set at 2, and was instead set at 1.8. In certain situations the μ_r would become greater than the critical value during the integration process and the algorithm would raise errors when trying to evaluate the $cos^{-1}()$ expression. When the goal was set at $[x = 0, y = -2]$ (or $[\theta = -\frac{1}{2}\pi, r = 2]$ in polar coordinates) this error occurred, for example. Interestingly, when this goal was switched to the right side ($[x = 0, y = 2]$) as is shown in figure 6.21, the error did not occur. Some small difference in the dynamics of the model caused the value of μ_r to remain below the critical value for the goal on the right side but not on the left side.

The implementation using polar coordinates and the law of cosines demonstrates that when the degrees of freedom are considered in the model, it is possible to split the representations of the end effector positions and the joint angles of a robot arm hierarchically, and to let these representations converge to correspond to a predefined goal introduced at the top level. It has also become clear from this section and the simulation in section 6.3.3 that some functions (in this case inverse trigonometric functions) can cause problems if they are not well-defined for all input values, and can prevent the simulation from converging on the desired solution. Another disadvantage of polar-coordinate model is that the second level $H2$ only contains variable representing the position of the end effector. It is therefore not possible to model desired behaviour based on the position of the elbow joint, for example if one were to try and avoid an obstacle.

[3]The simulation for the polar coordinate implementation was run with the same values for the precision matrices as in the previous simulations, with the learning rates being $\kappa_\mu = 1, \kappa_v = 1, \kappa_a = 10$, the values of the dynamic precisions Π_ω set at 1 and the sensory and causal precisions Π_z set at 5.

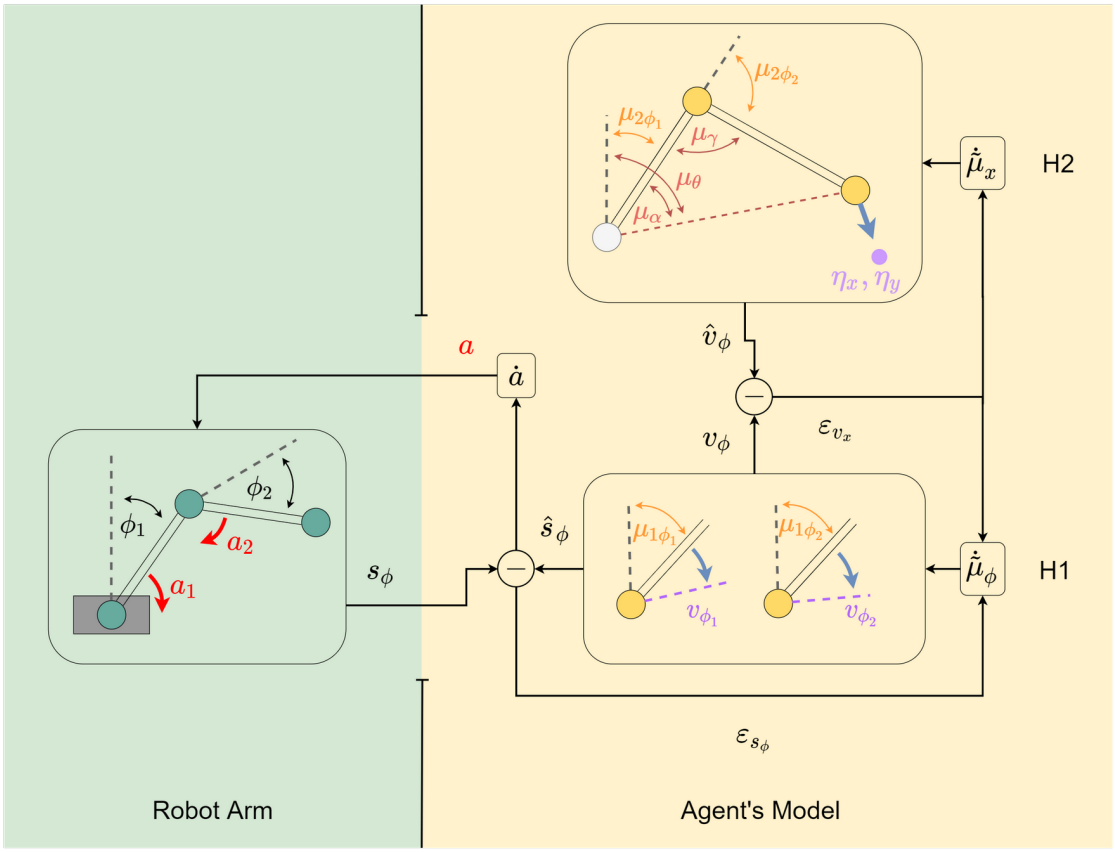


Figure 6.19: The robot arm and the internal model of the agent the causal states v_ϕ are predicted in $H2$ based on the law of cosines.

H1

$$f_1 = \begin{bmatrix} \hat{\mu}_{\phi_1'} \\ \hat{\mu}_{\phi 2'} \\ \hat{\mu}_{\phi 1''} \\ \hat{\mu}_{\phi 2''} \end{bmatrix} = \begin{bmatrix} (v_{\phi_1} - \mu_{\phi_1}) \\ (v_{\phi_2} - \mu_{\phi_2}) \\ -\mu_{\phi_1'} \\ -\mu_{\phi_2'} \end{bmatrix} \tag{6.14}$$

$$g_1 = \begin{bmatrix} \hat{s}_{\phi 1} \\ \hat{s}_{\phi 2} \\ \hat{s}'_{\phi 1} \\ \hat{s}'_{\phi 2} \end{bmatrix} = \begin{bmatrix} \mu_{\phi_1} \\ \mu_{\phi_2} \\ \mu_{\phi_1'} \\ \mu_{\phi_2'} \end{bmatrix} \tag{6.15}$$

H2

$$f_2 = \begin{bmatrix} \hat{\mu}_{\theta'} \\ \hat{\mu}_{r'} \\ \hat{\mu}_{\theta''} \\ \hat{\mu}_{r''} \end{bmatrix} = \begin{bmatrix} \theta_{goal} - \mu_\theta \\ r_{goal} - \mu_r \\ -\mu_{\theta'} \\ -\mu_{r'} \end{bmatrix} \tag{6.16}$$

$$g_2 = \begin{bmatrix} v_{\phi_1} \\ v_{\phi_2} \end{bmatrix} = \begin{bmatrix} \mu_\theta - cos^{-1}\left(\frac{L_2^2 - L_1^2 - \mu_r^2}{2 \cdot L_1 \cdot \mu_r}\right) \\ \pi - cos^{-1}\left(\frac{\mu_r^2 - L_1^2 - L_2^2}{(2 \cdot L_1 \cdot L_2)}\right) \end{bmatrix} \tag{6.17}$$

Figure 6.20: The equations producing the predictions by the agent with model 3, based on polar coordinates. The functions f produce predictions for the dynamic states $\hat{\boldsymbol{\mu}}$. At level $H1$, function g_1 produces predictions for the sensory states $\hat{\boldsymbol{s}}$. At level $H2$, function g_2 produces predictions for the causal states $\hat{\boldsymbol{v}}$.

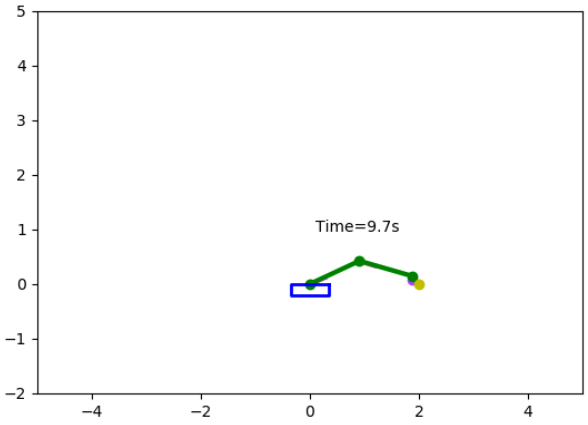


Figure 6.21: The robot arm using the polar coordinate model reaches the goal at $[x = 2, y = 0]$. Implementations with the goal at $[x = -2, y = 0]$ as in previous examples resulted in simulation errors.

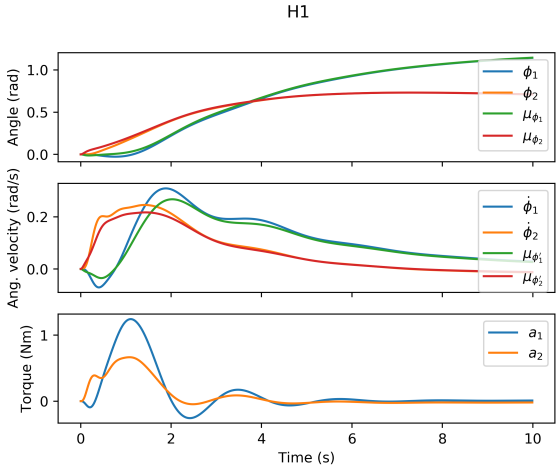


Figure 6.22: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$. $[a_1, a_2]$ are the action torques. The convergence of the angels becomes very slow as they get closer to the goal.

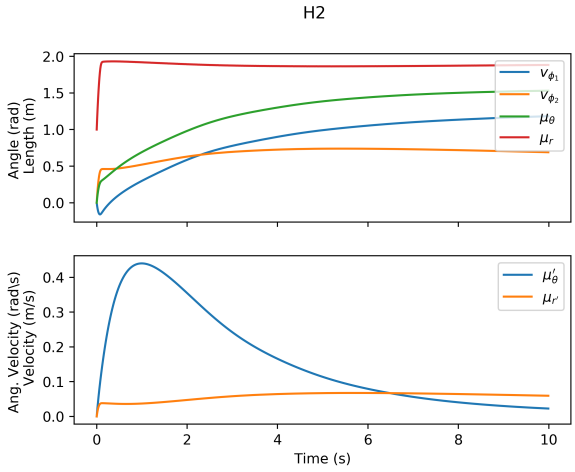


Figure 6.23: The causal states v_{ϕ_1}, v_{ϕ_2} in H_1 and the beliefs of the position of the end effector in polar coordinates μ_θ, μ_r in H_2.

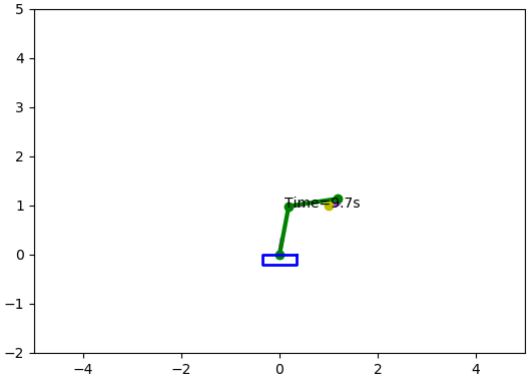


Figure 6.24: The robot arm using the polar coordinate model reaches the goal at $[x = 1, y = 1]$.

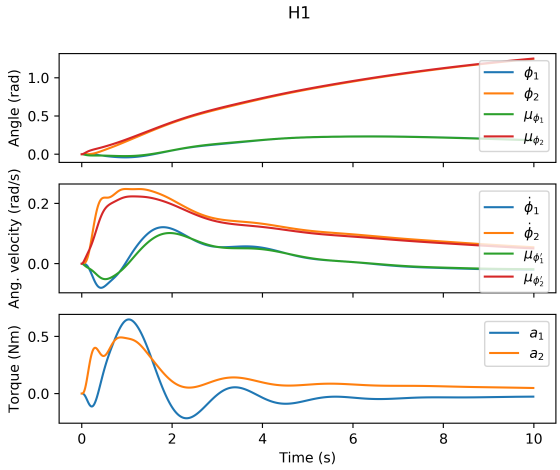


Figure 6.25: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$. $[a_1, a_2]$ are the action torques. The convergence of the angels becomes very slow as they get closer to the goal.

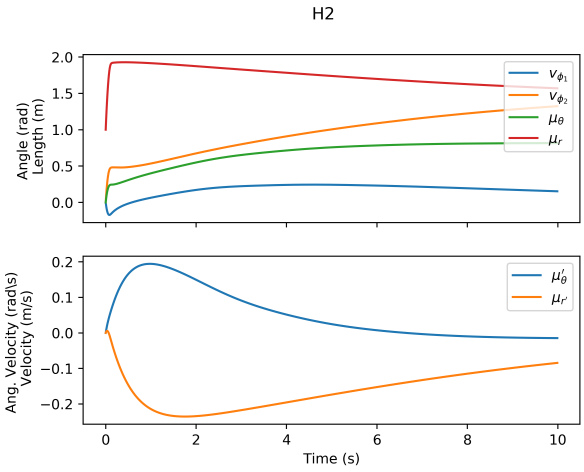


Figure 6.26: The causal states v_{ϕ_1}, v_{ϕ_2} in H_1 and the beliefs of the position of the end effector in polar coordinates μ_θ, μ_r in H_2.

6.4.3 Constrained attracting joint positions

A different, more direct way to make sure there are a lower number of degrees of freedom and to constrain the second level states to the appropriate configurations is to explicitly implement these constraints. The simulations in section 6.3 failed to reach the desired location because the dynamics in the internal model did not include information about the lengths of the robot arm segments, and therefore there were multiple joint positions that were valid for a given set of angles. The model discussed here considers these constraints by changing the dynamics of the beliefs about the elbow position μ_{x_a}, μ_{y_a}. Instead of assuming that the elbow joint has the same velocity as the end effector (as is done in section 6.3.1, the elbow joint is assumed to be attracted to the origin and the end effector, to within a certain distance. This is done by introducing $sign$ terms. For example, the $sign_0$ term (6.21) ensures the states representing the elbow joint (μ_{x_a}, μ_{y_a}) reflect the proper distance from the origin, by returning a positive value if the distance is larger, and a negative value if the distance is smaller. This value is used in the dynamic equations 6.22 so that the states (μ_{x_a}, μ_{y_a}) converge to a certain distance around the origin. An equilibrium is reached at a circle of radius L_1 around the origin. A similar behaviour is implemented for maintaining the distance L_2 with the end effector using the term $sign_1$. This attracting behaviour will maintain the proper constraints on the robot arm positions.

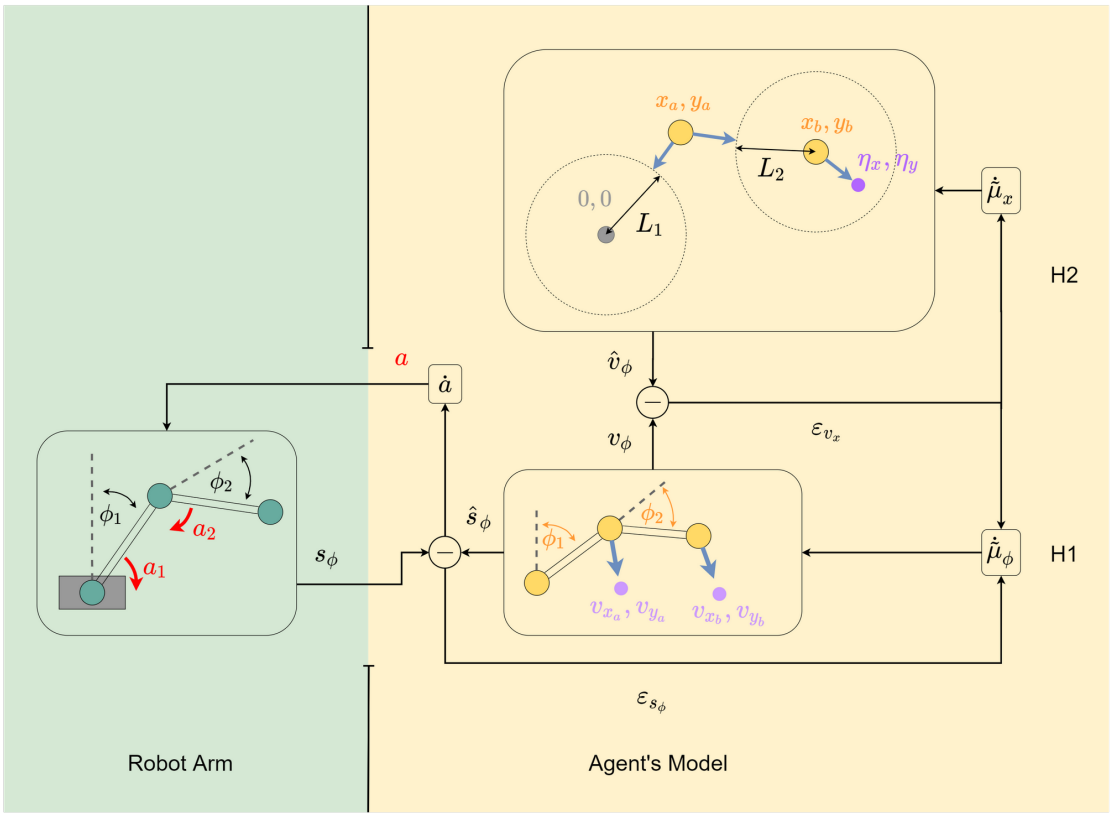


Figure 6.27: The robot arm and the internal model of the agent. the beliefs about joint positions $\mu_{(x,y)}$ in $H2$ are subject to constraints described as attracting velocities.

H1

$$
\begin{aligned}
x_1 &= sin(\mu_{\theta_1}) \cdot L_1 \\
y_1 &= cos(\mu_{\theta_1}) \cdot L_1 \\
x_2 &= sin(\mu_{\phi_1}) \cdot L_1 + sin(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2 \\
y_2 &= cos(\mu_{\phi_1}) \cdot L_1 + cos(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2
\end{aligned}
\tag{6.18}
$$

$$
f_1 = \begin{bmatrix} \hat{\mu}_{\phi'_1} \\ \hat{\mu}_{\phi'_2} \end{bmatrix} = \begin{bmatrix} (v_{x_a} - x_a) \cdot y_1 - (v_{y_a} - y_1) \cdot x_1 \\ (v_{x_b} - x_b) \cdot (y_2 - y1) - (v_{y_b} - y_2) \cdot (x_2 - x_1) \end{bmatrix}
\tag{6.19}
$$

$$
g_1 = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}'_1 \\ \hat{s}'_2 \end{bmatrix} = \begin{bmatrix} \mu_{\phi_1} \\ \mu_{\phi_2} \\ \mu_{\phi'_1} \\ \mu_{\phi'_2} \end{bmatrix}
\tag{6.20}
$$

H2

$$
\begin{aligned}
sign_0 &= \sqrt{\mu_{x_a}^2 + \mu_{y_a}^2} - L_1 \\
sign_1 &= \sqrt{(\mu_{x_b} - \mu_{x_a})^2 + (\mu_{y_b} - \mu_{y_a})^2} - L_2
\end{aligned}
\tag{6.21}
$$

$$
f_2 = \begin{bmatrix} \hat{\mu}_{x'_a} \\ \hat{\mu}_{y'_a} \\ \hat{\mu}_{x'_b} \\ \hat{\mu}_{y'_b} \end{bmatrix} = \begin{bmatrix} \mu_{x_a} \cdot sign_0 + (\mu_{x_b} - \mu_{x_a}) \cdot sign1 \\ \mu_{y_a} \cdot sign_0 + (\mu_{y_b} - \mu_{y_a}) \cdot sign1 \\ \eta_x - \mu_{x_b} \\ \eta_y - \mu_{y_b} \end{bmatrix}
\tag{6.22}
$$

$$
g_2 = \begin{bmatrix} \hat{v}_{x_a} \\ \hat{v}_{y_a} \\ \hat{v}_{x_b} \\ \hat{v}_{y_b} \end{bmatrix} = \begin{bmatrix} \mu_{x_a} \\ \mu_{y_a} \\ \mu_{x_b} \\ \mu_{y_b} \end{bmatrix}
\tag{6.23}
$$

Figure 6.28: The equations producing the predictions by the agent with model 4, implementing constraints at $H2$. The functions f produce predictions for the dynamic states $\hat{\boldsymbol{\mu}}$. At level $H1$, function g_1 produces predictions for the sensory states $\hat{\boldsymbol{s}}$. At level $H2$, function g_2 produces predictions for the causal states $\hat{\boldsymbol{v}}$.

6.4.4 Constrained Joints Results

Once more, the simulation is run for two situations: one with the end goal set at $[x = -2, y = 0]$ and one with the end goal set at $[x = 1, y = 1]$. Figure 6.29 and 6.32 show the configurations that the simulation converges to for those goals. The simulation for the $[x = -2, y = 0]$ goal was run for 20 seconds instead of ten as the robot arm converged to the goal very slowly. Even after almost 20 seconds the robot arm has not completely reached the outstretched position. This is probably because the rate at which the second level beliefs move towards their respective goals decreases linearly with the distance to those goals. As the goals are approached the attraction towards the goals diminish which cause the last part of the convergence to be very slow.

Despite the slow convergence, the algorithm is able to converge towards all the goals. This improves upon the implementation in section 6.4.1 by not having any functions that are not defined for certain inputs. This means that the simulation will not run into expressions that are invalid. Also, the second level now contains the positions for both the end effector and the elbow joint. This additional information could be used for obstacle avoidance, for example.

Although this method accomplishes the desired goal-reaching behaviour, this functionality could also be implemented relatively simply in a single level, as will be demonstrated in 6.5.1. A close look at $H1$ and $H2$ reveals that the information about the arm segment lengths L_1, L_2 is already included, in $H1$, so why should these constraints be implemented a second time in $H2$? This raises the question what the hierarchical representation of the agents model effectively achieves. It has become clear that it is possible to achieve goal-reaching behaviour with hierarchical differentiation between x, y coordinates and the joint angles ϕ, but this split does not seem to be necessary or even practical. If the goal reaching behaviour can be achieved with a single level, what effective dynamics might be described in a second level? The next section will discuss another, perhaps more practical functionality that could be assigned to the hierarchical model.

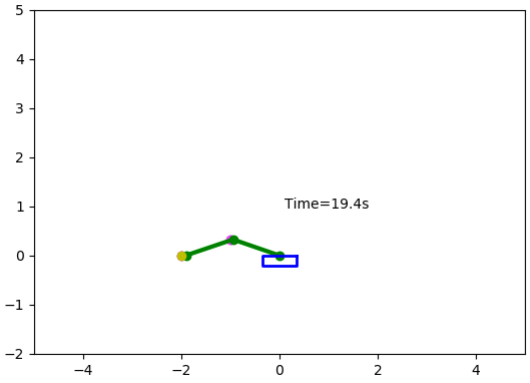


Figure 6.29: The robot arm using the constrained model reaches the goal at $[x = -2, y = 0]$, but converges more slowly as it nears the goal. Model 4 does not raise the computational errors that model 3 did.

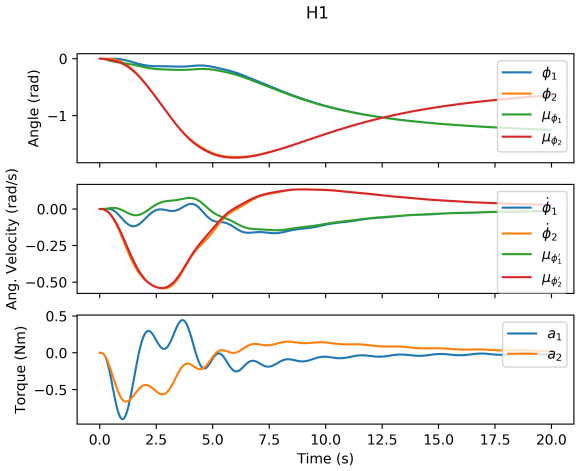


Figure 6.30: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$. $[a_1, a_2]$ are the action torques. The angles move towards the appropriate positions to reach the goal, albeit slowly.

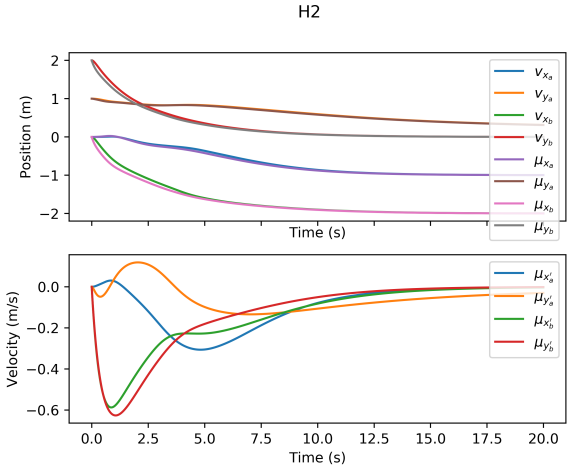


Figure 6.31: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2.

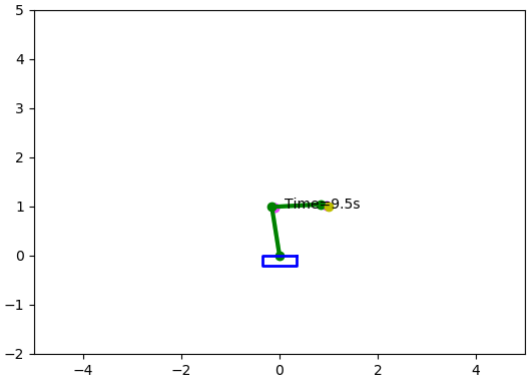


Figure 6.32: The robot arm using the constrained model reaches the goal at $[x = 1, y = 1]$, but converges more slowly as it nears the goal.

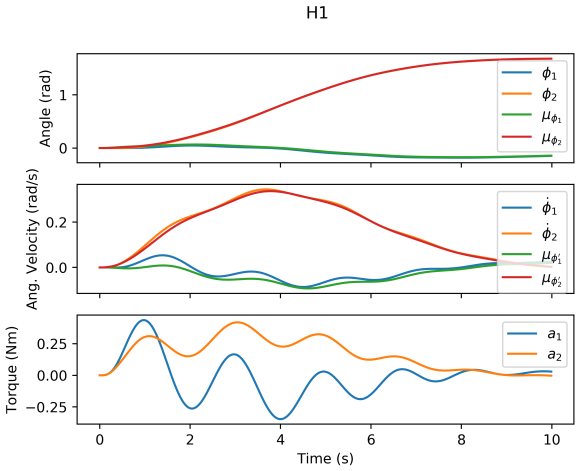


Figure 6.33: The real-world states $[\phi, \dot{\phi}]$ and the corresponding beliefs $[\mu_\phi, \mu_{\phi'}]$ at $H1$. $[a_1, a_2]$ are the action torques. The angles move towards the appropriate positions to reach the goal, albeit slowly.

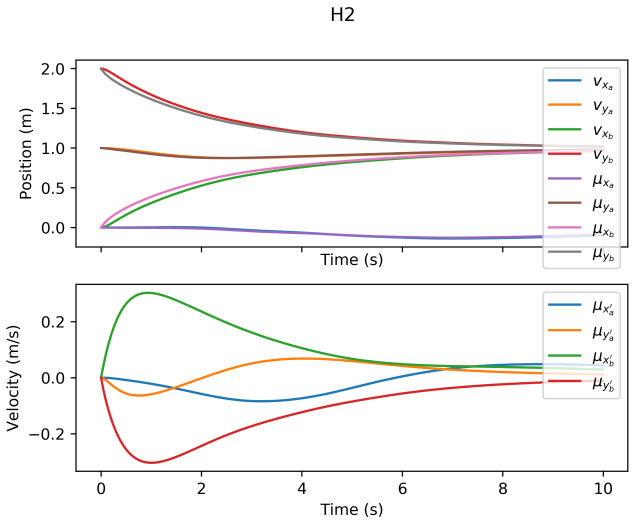


Figure 6.34: The causal states $v_{x,y}$ in H_1 and the beliefs of the positions $\mu_{(x,y)}, \mu_{(x',y')}$ in H_2.

6.5 Hierarchical Task Expansion

The previous sections have explored the separation of angular coordinates and x, y coordinates between two levels, $H1$ and $H2$. The reasoning behind this structure was that the positions of the elbow joint and end effector are a logical abstraction of the sensory states ϕ_1 and ϕ_2. The positions are not directly observable but can be deduced from the available sensory information. However, it became clear that constraints have to be considered when the second level involves more variables than the first level.

The task of the aforementioned algorithms can be condensed to a single level by simply adjusting the angles directly to a goal position, as implemented in equation 6.25. The function of H_1 then becomes finding joint angles that match the given goal position, and the problem concerning constraints is handled in this level. A second level need then only provide a desired end effector position, without having to involve these constraints. The function of H_2 could then be to dictate a path for the end effector. This approach effectively expands the complexity of the robot's task, and is explored in the following section.

6.5.1 Circle Path

The new functional structure for the hierarchical generative model is investigated by specifying a path for the end effector in the second level $H2$. The dynamics in $H1$ are described in such a way that the joint angles converge to a configuration that achieves a desired position for the end effector. In this example we are still considering 2 robot arm segments (L_1 and L_2), but in principle this approach could also work with a robot arm consisting of more segments. This is more consistent with the idea of hierarchy as it was approached in chapter 5, where lower levels contained more variables and more information.

H_2 now specifies a path of the end effector. It no longer involves the position of the elbow joint or any other considerations, and the details of achieving that position through the appropriate joint angles is left as a task for H_1. H_2 describes a point following a circular path that is in x, y coordinate space. A circular path was chosen because it is a repetitive behaviour that can be easily inspected. This path is also reasonably simple to implement by a vector field describing a limit cycle. This vector field simply assigns a velocity to every point in x, y space, such that a particle moving with these velocities will end up in a clockwise motion around a certain location.

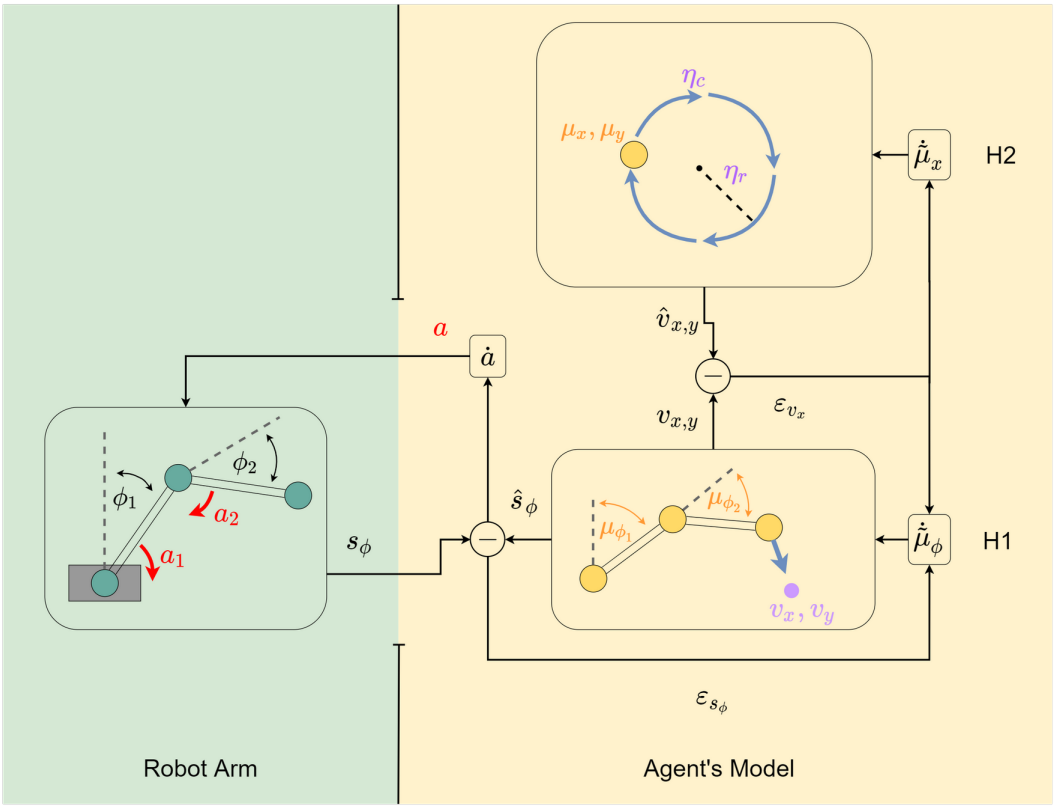


Figure 6.35: A different approach to organizing internal representations in a hierarchy. $H2$ describes a path for the end effector $\mu_{(x,y)}$, $H1$ must produce the appropriate joint angles μ_ϕ.

H1

$$\begin{aligned} x_1 &= sin(\mu_{\phi_1}) \cdot L_1 \\ y_1 &= cos(\mu_{\phi_1}) \cdot L_1 \\ x_2 &= sin(\mu_{\phi_1}) \cdot L_1 + sin(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2 \\ y_2 &= cos(\mu_{\phi_1}) \cdot L_1 + cos(\mu_{\phi_1} + \mu_{\phi_2}) \cdot L_2 \end{aligned} \tag{6.24}$$

$$f_1 = \begin{bmatrix} \hat{\mu}_{\phi'_1} \\ \hat{\mu}_{\phi'_2} \end{bmatrix} = \begin{bmatrix} (v_{x_1} - x_1) \cdot y_1 - (v_{y1} - y_1) \cdot x_1 \\ (v_{x_2} - x_2) \cdot (y_2 - y_1) - (v_{y2} - y_2) \cdot (x_2 - x_1) \end{bmatrix} \tag{6.25}$$

$$g_1 = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \\ \hat{s}'_1 \\ \hat{s}'_2 \end{bmatrix} = \begin{bmatrix} \mu_{\phi_1} \\ \mu_{\phi_2} \\ \mu_{\phi'_1} \\ \mu_{\phi'_2} \end{bmatrix} \tag{6.26}$$

H2

$$\begin{aligned} x &= \mu_x - o_x \\ y &= \mu_y - o_y \end{aligned} \tag{6.27}$$

$$f_2 = \begin{bmatrix} \hat{\mu}_{\phi'_1} \\ \hat{\mu}_{\phi'_2} \end{bmatrix} = \begin{bmatrix} c \cdot y + x \cdot (r^2 - x^2 - y^2) \\ -c \cdot x + y \cdot (r^2 - x^2 - y^2) \end{bmatrix} \tag{6.28}$$

$$g_2 = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \tag{6.29}$$

Figure 6.36: The equations producing the predictions by the agent with model 5, describing a circular motion in $H2$. The functions f produce predictions for the dynamic states $\hat{\boldsymbol{\mu}}$. At level $H1$, function g_1 produces predictions for the sensory states $\hat{\boldsymbol{s}}$. At level $H2$, function g_2 produces predictions for the causal states $\hat{\boldsymbol{v}}$.

6.5.2 Circle Path Results

The simulation was run with the new generative model, implementing a more complex task in the hierarchical levels. After some tuning [4], the robot arm was able to perform a circular motion with the end effector. This motion is described in $H2$ (6.36) by several parameters. $[o_x, o_y]$ specify the location of the center of the circle, r is the radius and c determines the speed at which the circular motion is executed. These parameters were set at the following values: $[o_x = 1, o_y = 1, r = 0.7, c = 2]$. The beliefs about the angles μ_ϕ at H_1 are adjusted to the beliefs about the end effector position $\mu_{x,y}$ at H_2 through the prediction error of the causal states $v_{x,y}$.

The prediction error feedback is not instantaneous and therefore the actual end effector will lag behind the believed end effector position. This lag means the robot arm traces a smaller circle than is dictated by the dynamics of the internal model. The feedback given by prediction errors cause the internal beliefs to be somewhat adjusted to this lag. This is why the dynamic states μ_x and μ_y also follow a slightly smaller circle than is dictated by the internal dynamics. Figure 6.39 shows those states following a path with roughly a radius of 0.5 around the point $[x = 1, y = 1]$, although the modelled dynamics describe a radius of $r = 0.7$. When the believed end effector position was closer to the origin as depicted in figure 6.37, the robot arm had more trouble following it and the lag was greater. This was probably because a faster adjustment of the joint angles was required in this configuration.

This simulated example shows an effective use of the hierarchical generative model for achieving desired behaviour in a robot arm. This example was different than previous implementations in that it used the second hierarchical level to describe the path of the end effector, rather than a static goal. This illustrates how hierarchical levels could be stacked to describe increasingly complex behaviour.

[4] The simulation of the circle path model was first run with the same parameter values as the previous simulations, $[\kappa_\mu = 1, \kappa_v = 1, \kappa_a = 10]$, 1 for the values of Π_ω and 5 for the values of Π_z. However, it became clear that the robot arm was lightly unstable for these values and would start to oscillate. Increasing the values of Π_z to 10 to make the agent adhere more to sensory feedback increased the stability. The reason for this is probably that the modelled dynamics at the first level have changed. The formula f_1 (6.25) no longer formulates angular velocities based on two expected joint positions, but rather depend only on the end effector position. Apparently modelling the dynamics in this way decreases the stability of the control feedback. This instability is probably a result of the fact that the modelled dynamics f_1 do not take into account factors like inertia or the forces in the joint. These dynamics are quite complicated and are not included in the expressions. As in the previous examples, f_1 only produces expectations on the angular velocities and leaves any inaccuracies in the prediction to be compensated by the action feedback that arises from the prediction error. It has become clear in this simulation and during the simulation of previous cases that this feedback loop is not necessarily stable.

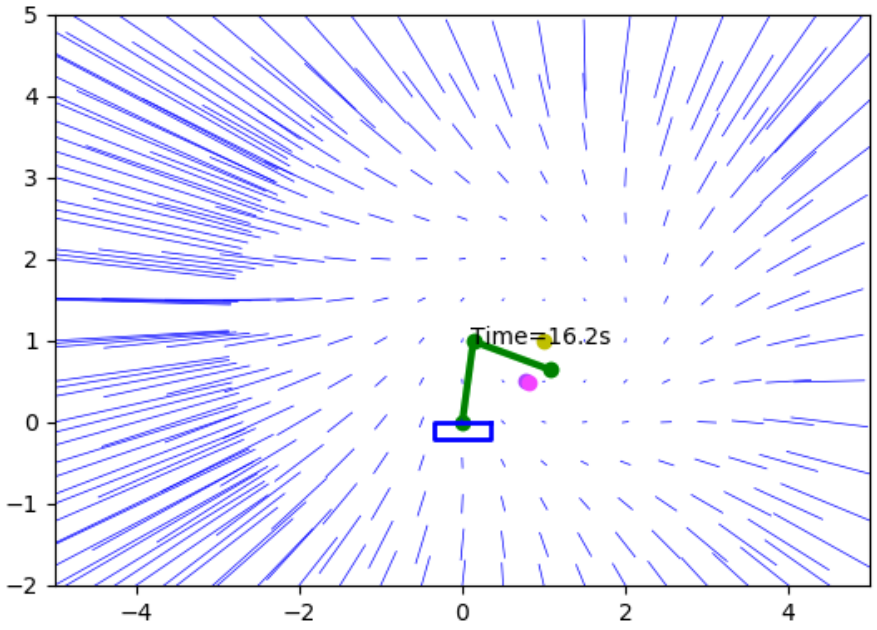


Figure 6.37: This image shows the robot arm following a goal position (μ_x, μ_y) represented by the pink dot. The goal position itself follows a circular path which is determined by the vector field described by the blue lines. This causes the robot arm to 'draw circles' around the yellow dot. The robot arm angles are modelled in the first level and the goal position and its dynamics in the second level.

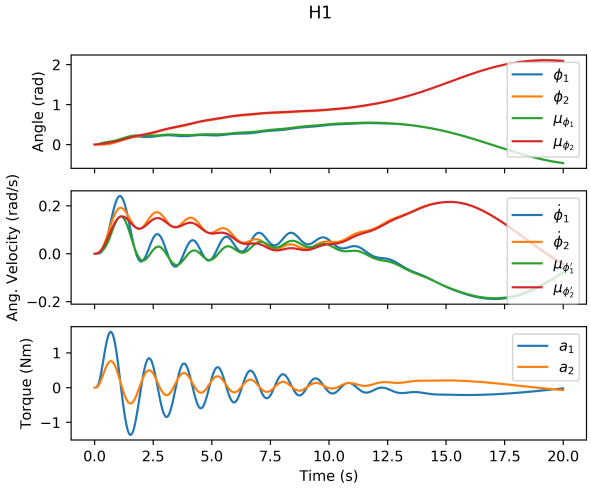


Figure 6.38: The robot arm angles and the derivatives and the corresponding beliefs of the agent at the first hierarchical level. The bottom graph shows the joint torques enacted by the agent.

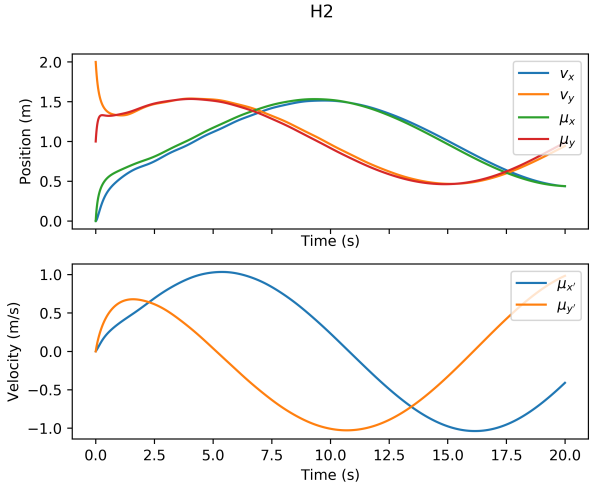


Figure 6.39: The second-level beliefs of the corresponding to the x and y location of the end effector. The end effector assumes a circular path around a specified point.

6.6 Chapter Summary

This chapter has examined several different versions of a hierarchical generative model to evoke specific behaviour in a 2-D robot arm with 2 segments. The first approach was to make a hierarchical distinction between joint positions and the joint angles, which could be compared directly with the sensory information. The joint positions are a logical abstraction or derivations from the joint angles. Desired dynamics of the joint positions, namely convergence towards a specific goal, were described in the second hierarchical level H_2 and the dynamics that would produce the appropriate joint angles were included in the first hierarchical level H_1. The predictions made by the internal model would enforce the desired behaviour in the (simulated) robot arm.

This approach raised a number of key issues in the application of hierarchical models for control purposes. One of the issues was the use of functions that are undefined or ill-defined for certain input values. Several trigonometric expressions used to describe geometric relationships in the agents model had this problem and prevented the simulation from running effectively.

The most important lesson was the fact that higher hierarchical levels could not contain more degrees of freedom than lower hierarchical levels. This is a logical consequence of the fact that it is impossible to extract more information about the outside world than is provided by the sensory signals, without some prior information or constraints predefined in the agent's model. It is possible to implement these constraints in the model, but a careful consideration of the degrees of freedom in the model is needed. This is an important concept to remember when constructing generative models for control or perception purposes with hierarchical active inference.

This insight lead to a reconsideration of the functional mechanisms of the hierarchical levels. It has been posited before (chapter 5) that an effective use of hierarchical models is to summarize or abstract information in higher levels. The term abstraction is sometimes used to describe the process of representing only essential information and leaving out unnecessary details, and this description is also appropriate for hierarchical models. A new example that better reflected this idea was explored in the last section (6.5.1). In this example, the details of the robot arm (the joint angles) were restricted to the first level and the second level dealt with the essential information extracted form these angles (end effector position) and describing its desired behaviour. This approach worked as expected and demonstrated a practical implementation of hierarchy in the internal model of an active inference agent. The results showed that the behaviour described by the agent's model and the performance of the robot arm still differed somewhat, and that unmodelled dynamics and delay in prediction error feedback are factors that should be considered for an effective hierarchical control model. The insights provided by these simulations provide a basis for more extensive studies into hierarchical active inference for robot control. The analysed cases highlight some important factors to consider in constructing hierarchical generative models for robotic agents and can be a useful reference for future work.

Chapter 7

Conclusion

This thesis has discussed and demonstrated a robot control algorithm based on hierarchical active inference. A series of simulations have shown that effective robot control is possible with this method and some of the potential pitfalls have been uncovered. These insights have been gained by pursuing the following research goals:

1. Present a concise description of the theoretical background of active inference.
2. Provide an explanation of how a hierarchical model can function within active inference.
3. Put a hierarchical active-inference based control algorithm into executable code for simulations, that can implement various internal and external models.
4. Demonstrate the algorithm for a single-level linear goal-reaching control problem.
5. Demonstrate the algorithm for a linear goal reaching control problem using a hierarchical generative model.
6. Apply the algorithm to a 2-D robot arm with various hierarchical generative models to investigate the function of the hierarchical model for active inference in robot control.

The main finding of this thesis is that hierarchical models can be applied to represent and achieve high-level goals for an active inference agent. These high-level goals generate additional complexity in the behaviour of the system. The success of this approach depends on the generative model that is constructed, and can be tuned through the precision parameters. For these goal reaching cases to achieve the desired solution it is crucial that that sensory information is summarized in higher hierarchical levels.

Active inference describes an inherent action-feedback system for biological or artificial agents. The hierarchical formulation of this theory inspires a hierarchical control method that could potentially facilitate complex behaviour. Chapter 2 explains that this theory is based on the minimization of a quantity called the free energy. Various assumptions such as the Laplace Approximation are needed for the agent to be able to evaluate the Free Energy quickly and simply. Taking into account these assumptions, the Free Energy is equivalent to the weighted squared prediction error of a model employed by the agent. This model can be divided into multiple hierarchical levels, in which each level generates predictions for causal states of the level below it. The prediction errors on these causal states invoke upward and downward signals between hierarchical levels.

The mechanism of the hierarchical active inference based algorithm has been demonstrated in various simulations, first for a single level and later with hierarchical examples. A position-reaching problem on a single cart has reiterated an approach introduced in earlier work [22] that establishes active inference as a method for robot control. By analysing the eigenvalues of the closed-loop system, the precisions Π and the learning rates κ could be adjusted to obtain a stable solution.

Generalized coordinates that introduce higher order states in the generative model were considered for this implementation. The evaluation of the eigenvalues suggested that the inclusion of these generalized coordinates had some effect on the stability, but the difference was negligible. Therefore, generalized coordinates do not seem to be relevant for simple and noiseless systems such as the single cart implementation.

A similar case with more carts and a hierarchical generative model was implemented in a second simulation. The algorithm proved to be effective in reaching high level goals that require the execution of lower level goals. In this process, higher levels summarize the sensory information provided to the agent in bottom-up signals, and actions to enforce higher level goals are activated by top-down signals. An additional simulation with a disabled cart has tested a situation in which the agent's belief is unreachable in the physical system. The results of this impaired system showed that the prediction error caused by discrepancy propagates up the hierarchical levels. At each level compromise is established between the agent's beliefs and the available information from lower levels.

Finally, these principles have been demonstrated in a practical robot-arm example. Through an exploratory investigation it has been determined that the hierarchical active inference process can be used to implement a goal-reaching task in a simulated robot arm. However, this requires a careful consideration of the generative model that is employed by the agent. For example, several failed attempts showed that expressions in the model must be well-defined for all reachable states. In many cases computational singularities were encountered that caused the simulation to stop or reach an incorrect solution.

One of the most important outcomes that the different robot arm cases demonstrated was that higher hierarchical levels cannot contain more independent states than the lower levels. In the cases discussed in this report this meant that two sets of x, y coordinates could not be accurately derived from two joint angles ϕ unless they were made dependant (by implementing constraints). This is an important consideration for hierarchical models and reflects the fact that the agents model cannot contain more information than is available through the sensory states.

Deriving positions from joint angles turned out not to be a suitable task to apply to a hierarchical model because of the issue discussed above. Hierarchical divisions make most sense when redundancies can be omitted and information can be summarized in higher level representations. Therefore, a more sensible approach was to only model the behaviour of the end effector in the higher level. The new implementation of the hierarchical model succeeded in making the robot arm follow a circular path, instead of simply reaching a goal. This final example demonstrated the fact that hierarchical levels can be used to add complexity to a robot's task.

In conclusion, this thesis has presented a versatile method for robot control that incorporates the concept of hierarchical active inference. It has been demonstrated in multiple examples and used to achieve controlled behaviour in a simplified robot arm that utilizes the hierarchical structure effectively. The investigation has touched on many subjects that call for further research, but the hierarchical implementation of active inference is introduced as a promising approach to achieving complex tasks in robotic systems.

Chapter 8

Discussion

This thesis has presented and demonstrated a control method for continuous systems based on active inference and the underlying Free Energy Principle. The Free Energy Principle itself is a very broad theoretical framework that encompasses many different types of inference and control methods. The control method in this work represents a very specific interpretation of this theory, and includes many assumptions and aspects that can be expanded or improved upon.

8.1 Assumptions

One of the most important restrictions that have been adopted is that the state updates have been limited to dynamic states. Other elements such as the model parameters or the Precisions Π were fixed. In much of the work by Karl Friston on the Free Energy principle a distinction is made between states or parameters that change relatively rapidly and those that change more slowly or remain static. The DEM (dynamic expectation maximization) process that he describes includes updating all of these parameters. [9]

The work presented here has focused on dynamic, continuous systems, relying on a state space representation of the world. An important and interesting aspect of active inference for continuous systems is the use of generalized coordinates. However, this aspect of active inference has not enjoyed the focus of this thesis. A short analysis of the implementation of higher orders in the single cart example in chapter 4 showed that the inclusion of higher orders some effect, but it was negligible. Generalized coordinates are most relevant in filtering and state estimation problems where the random fluctuations have a certain smoothness [9], [10]. It has been suggested that these higher order representations provide an advantage in state estimation by modelling the temporal correlation in the noise. However, noise has been omitted in the simulations that are discussed in this report, and the generalized coordinates are therefore less relevant.

Noise terms were omitted to enable a clearer analysis of the properties of the control method. Inclusion of noise would have made it more difficult to demonstrate the mechanism of the active inference approach to control. It was also unclear how dynamic noise should be included in the system. First of all, there is the issue of how to generate a random but temporally correlated disturbance. One possibility is to execute a Gaussian convolution of white noise, which produces a smooth but random signal. This possibility was partially explored and even included in the code. A second ambiguous issue is the introduction of the disturbances to the system and the representation in the generative model. A logical approach would be to include the disturbances as forces or torques in the simulation of the physical system, affecting only the accelerations (2nd dynamical order). However, the notion of generalized coordinates dictates that these disturbances are represented across all the dynamical orders. It it is still unclear how the derivatives of these disturbances should be derived to accurately represent smooth noise in a physical system, and due to time constraints this implementation was left out of this investigation. This means that no form

of noise was implemented in the simulations and so we are unable to evaluate the robustness of the control methods under disturbances.

Another interesting point of discussion regarding noise is that the precisions Π are related to the assumed presence of noise in the system. The precision parameters quantify the uncertainty that the agent has about the states due to the disturbances. However, during this investigation we have mostly discussed models that include states that are not present in the physical system, for example the goals in higher hierarchical levels. The precisions related to these states therefore could not be related to a physical disturbance in the real world. Rather, they represent the uncertainty that the agent has about the accuracy of these states in predicting the development of the physical system.

The simulations show that the internal states of the agent are not adapted instantly to the available sensory information. The path following behaviour of the last simulation illustrated this as the end effector trailed behind the expected position. The informational delay between levels suggests that control actions that require rapid feedback (for example to ensure stability) should be implemented in lower hierarchical levels, and dynamics that span longer periods of time can be implemented in higher levels. This is illustrated clearly in the circle-tracing robot arm, as the more general behaviour of a circular motion by the end effector expressed in the second level $H2$ is only accurate if the dynamics of the first level $H1$ are sufficiently fast.

The informational delay could also be related to the fact that in this thesis the higher order representations of the causal states was disregarded 2.21. It was assumed that the causal states were relatively static and that therefore the representation of higher orders was not necessary. However, in a hierarchical context with moving goal positions as in 6, the derivatives could provide useful information about the path of the causal states to lower levels. Perhaps the observed lag of the end effector of the robot arm would have been reduced had these derivatives been included.
An important issue from a control perspective is the generation of the actions. The actions are applied based on the prediction error between the predicted and the observed states. The action updates rely on the assumption that these actions have a straightforward and linear relation to the sensory observations, through the matrix $\boldsymbol{R}$. However, this is obviously not always the case. In the robot arm for example, the elbow joint torque is assumed to affect the elbow joint angle and the angular velocity directly and proportionally. In reality however, the effect is on the joint's angular acceleration, and also heavily dependant on the arm configuration. The generated actions therefore cannot optimally reduce the sensory prediction error. The matrix $\boldsymbol{R}$ can be adjusted to improve the response but in general the generated actions will provide sub-optimal control [3].

The results presented in chapter 6 indicate that a hierarchical organization of a predictive model is only effective when higher levels contain less independent states than lower levels. This is a logical consequence of the fact that more information can not be produced from limited sensory input. However, this is only true for any particular instance in time. It should be possible for an agent to deduce complicated properties of the external world through limited sensory signals given enough time. This is evident to us as human beings as we have an extensive understanding of a rich and varied world, but only only have limited sensory experiences at any given time. This does require a form of memory which is not present in the system as described in this thesis.

8.2 Future Work

It has been mentioned before that the free energy theory encompasses a wide range of inference and control methods. Considering the assumptions that have been made to arrive at the control algorithm in this thesis, it is possible that this approach is very similar or even equivalent to other, more conventional control methods. There are certainly other model based prediction processes that are used for state estimation and control. However, it is still unclear how these relate to the active-inference based control method, especially in a hierarchical context. It is also possible that the similarity to other control methods depends on the form of the generative model that is employed by the agent. Further work comparing the active inference approach to well-established control techniques could provide useful insights into issues such as stability and tuning, which have been more extensively researched in those domains.

The focus of this thesis was on the hierarchical implementation of the predictive model employed by the agent. Now that the mechanism of this hierarchical approach for robotic control has been demonstrated, the hierarchical models can be explored further. Hierarchy offers many different possibilities for predictive models. In this investigation the hierarchical structure has primarily been used to furnish objectives for the robotic agents. This was a convenient way of describing simple tasks to be accomplished. However, the hierarchical structure could also be used to represent the causes of complex sensory observations, or abstract concepts that summarize developments in the external environment of the agent.

To simplify this investigation into alternative functions for the hierarchical structure, it would initially be convenient to omit the action generation part of the process. Actions that minimize prediction error are an essential concept in active inference, however it would probably be useful to first analyse more complex hierarchical models as an inference mechanism before adding elements of action inputs to the system. One of the possibilities that can be explored is combining sensory information from different sources hierarchically to create a more accurate representation of the environment. Some work on combining sensory information using predictive coding has been done in [18], and [17] discusses inference using hierarchical dynamical models.

An interesting topic that is related to the idea of hierarchical inference is misinterpretation of presented sensory data, in other words illusions [5], [16]. If there are multiple causes that could result in a certain set of sensory information, an inference agent may arrive at the wrong conclusion. A simple example that could be used as a preliminary investigation is the silhouette of a rotating object. Without any depth cues, the 2-D projection of a rotating object offers no definitive information to determine which way the object is rotating. It would be interesting to investigate how an (active) inference agent handles ambiguous sensory signals.

In short, this thesis has produced many questions about active inference as a method for control, the assumptions that are made to implement the underlying Free Energy principle and the role of predictive models and hierarchy in robotics. Some insight has been gained in the vast possibilities of hierarchical active inference, but much more research on different aspects of the topic is warranted to produce effective motion control and learning strategies for robotics.

Bibliography

[1] Rick A. Adams, Stewart Shipp, and Karl J. Friston. Predictions not commands: Active inference in the motor system. *Brain Structure and Function*, 218(3):611–643, 2013.

[2] Bhashyam Balaji and Karl Friston. Bayesian state estimation using generalized coordinates. *Signal Processing, Sensor Fusion, and Target Recognition XX*, 8050:80501Y, 2011.

[3] Manuel Baltieri and Christopher L. Buckley. PID control as a process of active inference with linear generative models. *Entropy*, 21(3), 2019.

[4] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[5] Harriet Brown and Karl J. Friston. Free-energy and illusions: The Cornsweet effect. *Frontiers in Psychology*, 3(FEB):1–13, 2012.

[6] Christopher L. Buckley, Chang Sub 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.

[7] Daniel Camilleri and Tony Prescott. Analysing the limitations of deep learning for developmental robotics. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10384 LNAI:86–94, 2017.

[8] Daniel J Felleman and David C Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1:1–47, 1991.

[9] K. J. Friston, N. Trujillo-Barreto, and J. Daunizeau. DEM: A variational treatment of dynamic systems. *NeuroImage*, 41(3):849–885, 2008.

[10] Karl Friston. Hierarchical models in the brain. *PLoS Computational Biology*, 4(11), 2008.

[11] Karl Friston. What is optimal about motor control? *Neuron*, 72(3):488–498, 2011.

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

[13] Karl Friston, Jérémie Mattout, Nelson Trujillo-Barreto, John Ashburner, and Will Penny. Variational free energy and the Laplace approximation. *NeuroImage*, 34(1):220–234, 2007.

[14] 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.

[15] Sherin Grimbergen. *The state space formulation of active inference (Msc Thesis)*. 2019.

[16] Nina Alisa Hinz, Pablo Lanillos, Hermann Mueller, and Gordon Cheng. Drifting perceptual patterns suggest prediction errors fusion rather than hypothesis selection: Replicating the rubber-hand illusion on a robot. *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2018*, pages 125–132, 2018.

[17] Stefan J. Kiebel, Jean Daunizeau, and Karl J. Friston. A hierarchy of time-scales and the brain. *PLoS Computational Biology*, 4(11), 2008.

[18] Pablo Lanillos and Gordon Cheng. Adaptive Robot Body Learning and Estimation Through Predictive Coding. *IEEE International Conference on Intelligent Robots and Systems*, pages 4083–4090, 2018.

[19] M. Marsel Mesulam. From sensation to cognition. *Brain*, 121(6):1013–1052, 1998.

[20] Beren Millidge. Combining Active Inference and Hierarchical Predictive Coding: A Tutorial Introduction and Case Study, 2019.

[21] Guillermo Oliver, Pablo Lanillos, and Gordon Cheng. Active inference body perception and action for humanoid robots, 2019.

[22] Corrado Pezzato, Riccardo Ferrari, and Carlos Hernandez. A Novel Adaptive Controller for Robot Manipulators based on Active Inference, 2019.

[23] Léo Pio-Lopez, Ange Nizard, Karl Friston, and Giovanni Pezzulo. Active inference and robot control: a case study. *Journal of The Royal Society Interface*, 13(122):20160616, 2016.

[24] Rajesh P.N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79–87, 1999.

[25] Arend Schwab and Guido Delhaes. Lecture Notes Multibody Dynamics B, wb1413. 2009.

Appendices

Appendix A

Code

A.1 Main File

```
import numpy as np
import sympy as sym
from numpy import r_
from scipy.integrate import solve_ivp
from sympy import Matrix
from os import path

from physics import physics_cartgroups
from noise import noise_generator
from state_updates import solve_states
from generative_model import genmod_cartgroups

from graphics.graphics_cartgroups import drawgraphs_cartgroups_HAI
from graphics.graphics_cartgroups import animate_cartgroups

# Cart control with active inference
# Author: Victor van Vucht
# Date: 10 December 2020

# This script implements Hierarchical AI by combining linear generative models for several levels.

filename = 'Cart_Group_3levels_1st_order'     #Filename for saving mp4 and solution data

# Time parameters
t_end = 20
dt = 0.01
time_span = np.r_[0:t_end:dt]
symbolic = False        # allows the calculation of the update steps in symbolic variables
order = 'first'         # options: 'first', 'second', 'third'
eigenvalues = False

if symbolic == False:
    kmu = 2
    kv = 10
    ka = 10
else:
    kmu = sym.symbols('kmu')
    kv = sym.symbols('kv')
    ka = sym.symbols('ka')
```

```
40
41 learning_rates = [kmu, kv, ka]
42
43 print('Generative Process')
44 # Generative process Parameters & Variables
45 params, dx, ext_states, ext_states_init, noise_vars, dyn_order_process = physics_cartgroups.
       cart_groups(symbolic)
46
47 # Construct Noise
48 print('generate smooth noise')
49 noise = noise_generator.generate_noise(ext_states, dyn_order_process, t_end, dt) # Noise
       interpolation function, and arrays of the noises
50
51 #Construct generative model
52 print('Generative model')
53
54 gen_mod, int_states, int_states_init, prior = genmod_cartgroups.makemodel(params, ext_states,
       symbolic, order, eigenvalues)
55 # Belief of the effect of actions on the sensory input
56
57 states = ext_states + int_states
58 states_init = ext_states_init + int_states_init
59
60 solution = solve_states.solve(t_end, dt, dx, params, gen_mod, states, states_init, learning_rates,
        noise_vars, noise, symbolic, filename)
61
62 print('Animate')
63
64 drawgraphs_cartgroups_HAI(solution, dt, noise, states, FileName=filename, savegraph=True)
65 animate_cartgroups(solution, dt, params, prior, gen_mod, ghost=True, FileName=filename, savemovie=
       True)
```

A.2 Physics Module

```
import numpy as np
from numpy import diag
from numpy import array
from numpy import pi
from sympy import Matrix
from sympy import symbols

def cart_groups(symbolic):
    M1 = 1  #Kg,         Mass of the first arm segment
    M2 = 1
    M3 = 1
    M4 = 1
    M5 = 1
    M6 = 1
    M7 = 1
    M8 = 1
    M = [M1, M2, M3, M4, M5, M6, M7, M8]
    R = 1               #Friction

    dyn_order_process = 2

    params = [M, R]
    x = Matrix(symbols('x1 x2 x3 x4 x5 x6 x7 x8'))
    dx = Matrix(symbols('dx1 dx2 dx3 dx4 dx5 dx6 dx7 dx8'))
    states = Matrix([x, dx])

    s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16 = symbols('s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16')
    s = Matrix([s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15, s16])

    actions = Matrix(symbols('a1 a2 a3 a4 a5 a6 a7 a8'))

    M = Matrix(diag(M))
    G = -R*M*dx           # Friction forces
    ddx = M.LUsolve(G)  # can be used for linearization for the generative model

    dynamic_noise_forces = Matrix(symbols('T1_noise T2_noise T3_noise T4_noise T5_noise T6_noise T7_noise T8_noise'))  # external forces are a result of noise
    observation_noise = Matrix(symbols('s1_noise s2_noise s3_noise s4_noise s5_noise s6_noise s7_noise s8_noise s9_noise s10_noise s11_noise s12_noise s13_noise s14_noise s15_noise s16_noise'))

    active_ddx = M.LUsolve(
        G + dynamic_noise_forces + np.diag(array([0,1,1,1,1,1,1,1])) @ actions)  # M/(G+a) Calculate response of an actuated system (actual case)

    dstates = Matrix([dx, active_ddx])
```

```
43
44      noises = [dynamic_noise_forces, observation_noise]
45
46      ext_states = [states, s, actions]
47
48      # Initial conditions
49      x0 = array([-20, -17, -10, -3, 2, 4, 18, 20, 0., 0., 0., 0., 0., 0., 0., 0.])  # Initial
        states
50      s0 = array([-20, -17, -10, -3, 2, 4, 18, 20, 0., 0., 0., 0., 0., 0., 0., 0.])  # Initial
        observations
51      a0 = array([0, 0, 0, 0, 0, 0, 0, 0])  # Initial actions
52
53      ext_states_init = [x0, s0, a0]
54
55      return params, dstates, ext_states, ext_states_init, noises, dyn_order_process
```

A.3 Noise

```
1  import numpy as np
2  from numpy import r_
3  import matplotlib.pyplot as plt
4  from scipy.ndimage import gaussian_filter1d
5  from scipy.interpolate import interp1d
6
7  def generate_noise(ext_states, dyn_order_process, t_end, dt):
8      [x,s] = ext_states[:2]
9      num_x = len(x) // dyn_order_process  # Number of zero-order states
10     num_s = len(s)  # Number of sensory observations
11
12     mean_w = np.zeros(num_x)
13     mean_z = np.zeros(num_s)
14     covariance_w = np.diag(np.ones(num_x)) * 0  # covariance matrix of the dynamic noise
15     covariance_z = np.diag(np.ones(num_s)) * 0  # covariance matrix of the observation noise
16     smoothness_w = np.ones(num_x) * 3  # variance of the autocorrelation functions
17     smoothness_z = np.ones(num_s) * 3
18     noise_w = smoothnoise(mean_w, covariance_w, t_end, dt, smoothness_w, 'Dynamic Noise',
       plot_noise=False)[0]
19     noise_z = smoothnoise(mean_z, covariance_z, t_end, dt, smoothness_z, 'Observation Noise',
       plot_noise=False)[0]
20     noise_wz = r_[noise_w, noise_z]
21     time_span_noise = np.r_[0:noise_w.shape[1] * dt:dt]
22     interp_noise = interp1d(time_span_noise, noise_wz)
23     noise = [interp_noise, noise_w, noise_z]
24
25     return noise
26
27
28 def smoothnoise(mean, covariance, t_end, dt, smoothness, title, plot_noise=False):
29     #Function taking arguments for the scale, end time and smoothness of the noise
```

```
    # and the embedding order of the generalized states on which the noise is applied

    time_span = np.r_[0:t_end+1:dt]          # The ODE solver will access values larger than t_end
    , so add 1

    #Convolution Kernel

    stand_dev_kernel = np.sqrt(smoothness)
    kernel_range = 4 * max(stand_dev_kernel)    # Take the range as the 4 time the largest
    standard deviation of any of the kernels
    domain = np.r_[-kernel_range:kernel_range:dt]
    kernels = np.zeros((len(smoothness), len(domain)))

    for i in range(len(stand_dev_kernel)):
        z = -0.5 * (domain / stand_dev_kernel[i] - mean[i]) ** 2         #exponential of the
    Kernel function
        c = (1 / (stand_dev_kernel[i] * np.sqrt(2 * np.pi)))   #constant of the Kernel function
        kernels[i, :] = np.sqrt(dt)*c*np.exp(z)                 #dt is to compensate for the
    scaling with increased number of time steps

    #White gaussian noise
    trim = int(max(stand_dev_kernel) * 4 / dt)
    gauss_noise = np.random.multivariate_normal(mean, cov=covariance, size=[time_span.shape[0]+2*(
    trim)]).T


    #Smooth noise by gaussian filtering
    smooth_noise = np.zeros(gauss_noise.shape)

    for i in range(len(gauss_noise)):
        smooth_noise[i, :] = gaussian_filter1d(gauss_noise[i], stand_dev_kernel[i], mode='constant
    ', cval=0.0)


    #Trim the original noise

    gauss_noise = gauss_noise[:, trim:-trim]
    smooth_noise = smooth_noise[:, trim:-trim]


    if plot_noise:
        plotnoise(smooth_noise, gauss_noise, kernels, time_span, title)

    return smooth_noise, gauss_noise, kernels


def plotnoise(smooth_noises, gauss_noises, kernels, time_span, title):
    smooth_noises = smooth_noises.T
    gauss_noises = gauss_noises.T
    kernels = kernels.T
```

```
    time_span = np.resize(time_span, gauss_noises.shape)
    fig_noise, ax = plt.subplots(2, 2, sharey=True, num=2)
    fig_noise.suptitle(title, fontsize=12)
    ax[0, 0].plot(time_span, smooth_noises)
    ax[0, 0].set_title('Smoothed Noises')
    ax[0, 1].plot(kernels)
    ax[0, 1].set_title('Kernel')
    ax[1, 0].plot(time_span, gauss_noises, linewidth=0.1)
    ax[1, 0].set_title('Original Rough noises')
    plt.draw()
```

A.4 Generative Model

A.4.1 Construct Model

```
import numpy as np
from numpy import pi, r_, array
from scipy.linalg import block_diag
from sympy import Matrix, symbols

from . import models as mods

def makemodel(params, states, symbolic, order, eigenvalues):
    print('Construct generative model')

    H1 = mods.cart_groups_H1(params, symbolic)
    H2 = mods.cart_groups_H2(params, symbolic)
    H3 = mods.cart_groups_H3(params, symbolic)
    [f1, g1, mu1, v1, mu1_init, v1_init, precision_w1, precision_z1, num_states_1] = H1
    [f2, g2, mu2, v2, mu2_init, v2_init, precision_w2, precision_z2, num_states_2] = H2
    [f3, g3, mu3, v3, mu3_init, v3_init, precision_w3, precision_z3, num_states_3, prior] = H3

    f = Matrix([f1, f2, f3])
    g = Matrix([g1, g2, g3])
    mu = Matrix([mu1, mu2, mu3])
    v = Matrix([v1, v2, v3])

    mu_init = r_[mu1_init, mu2_init, mu3_init]
    v_init = r_[v1_init, v2_init, v3_init]

    D1 = np.diag(np.ones(len(mu1) - num_states_1), num_states_1) # Construct the "derivative
    operator" D matrix
    D2 = np.diag(np.ones(len(mu2) - num_states_2), num_states_2)
    D3 = np.diag(np.ones(len(mu3) - num_states_3), num_states_3)
    D_mu = block_diag(D1[0:len(f1),:], D2[0:len(f2),:], D3[0:len(f3),:])
    D = block_diag(D1, D2, D3)

    precision_w = np.diag(r_[precision_w1, precision_w2, precision_w3])
    precision_z = np.diag(r_[precision_z1, precision_z2, precision_z3])
```

```
34
35      x, a = states[0], states[2]
36      s_function = mods.sfunction_cartgroups(a,x)
37
38      gen_mod = [f, g, precision_w, precision_z, D_mu, D, s_function]
39      int_states = [mu, v]
40      int_states_init = [mu_init, v_init]
41
42      return gen_mod, int_states, int_states_init, prior
```

A.4.2 Models

```
1  import numpy as np
2  import sympy as sym
3  from numpy import pi
4  from numpy import array
5  from sympy import Matrix
6  from sympy import symbols
7
8  def cart_groups_H1(params, symbolic):
9      num_states = 8  # The number of states (excluding higher orders) in this level
10     mu_x = Matrix(symbols('mu_x1, mu_x2, mu_x3, mu_x4, mu_x5, mu_x6, mu_x7, mu_x8'))  # Zero order
       states
11     mu_dx = Matrix(symbols('mu_dx1, mu_dx2, mu_dx3, mu_dx4, mu_dx5, mu_dx6, mu_dx7, mu_dx8')) #
       First order states
12
13
14     mu = Matrix([mu_x,mu_dx])
15
16     mu_init = array([-20, -17, -10, -3, 2, 4, 18, 20, 0., 0., 0., 0., 0., 0., 0., 0.])  # Initial
       values
17
18     c = 2 #symbols('c') # convergence constant
19     v = Matrix(symbols('v_xg1, v_xg2, v_xg3, v_xg4'))
20     v_init = array([-15,-7,2,10])
21     goal = Matrix([v[0], v[0], v[1], v[1], v[2], v[2], v[3], v[3], 0, 0, 0, 0, 0, 0, 0, 0])
22
23     C = np.diag([-c]*len(mu_x))
24     Z = np.diag([0]*len(mu_x))
25
26     K = np.block([[C,Z],
27                   [Z,C,]])
28
29     f = c*goal + Matrix(K @ mu[0:16,:])
30     g = mu[0:16,:]
31     precision_w = array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1])
32     precision_z = array([10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10])
33     return f, g, mu, v, mu_init, v_init, precision_w, precision_z, num_states
34
35 def cart_groups_H2(params, symbolic):
```

```
    num_states = 4  # The number of states (excluding higher orders) in this level
    mu_g = Matrix(symbols('mu_xg1, mu_xg2, mu_xg3, mu_xg4'))
    mu_dg = Matrix(symbols('mu_dxg1, mu_dxg2, mu_dxg3, mu_dxg4'))

    mu = Matrix([mu_g,mu_dg])
    mu_init = array([-15,-7,2,10,0,0,0,0])  # np.array Initial values

    v = Matrix(symbols('v2_g1, v2_g2'))
    v_init = array([-17,17])
    goal = Matrix([v[0], v[0], v[1], v[1], 0, 0, 0, 0])

    c =  2 #symbols('c') #convergence constant

    C = np.diag([-c]*len(mu_g))
    Z = np.diag([0]*len(mu_g))

    K = np.block([[C,Z],
                  [Z,C,]])

    f = c*goal + Matrix(K @ mu[0:8,:]) #Careful not to apply double negatives, the sign is already
     in K
    g = mu[0:4,:]
    precision_w = array([1,1,1,1,1,1,1,1])
    precision_z = array([10,10,10,10])

    return f, g, mu, v, mu_init, v_init, precision_w, precision_z, num_states

def cart_groups_H3(params, symbolic):
    num_states = 2  # The number of states (excluding higher orders) in this level
    mu_g = Matrix(symbols('mu2_xg1, mu2_xg2'))
    mu_dg = Matrix(symbols('mu2_dxg1, mu2_dxg2'))


    mu = Matrix([mu_g,mu_dg])
    mu_init = array([-17,17,0,0])  # np.array Initial values

    v = Matrix([])
    v_init = array([])

    prior = 5
    goal = Matrix([prior,prior,0,0])

    c = 2 #symbols('c') # convergence constant

    C = np.diag([-c]*len(mu_g))
    Z = np.diag([0]*len(mu_g))

    K = np.block([[C,Z],
                  [Z,C,]])

```

```
    f = c*goal + Matrix(K @ mu[0:4,:])
    g = mu[0:2,:]
    precision_w = array([1,1,1,1])
    precision_z = array([10,10])
    return f, g, mu, v, mu_init, v_init, precision_w, precision_z, num_states, prior


def sfunction_cartgroups(actions,x):
    Rx = np.diag([1]*int(len(x)/2))
    Rdx = np.diag([1]*int(len(x)/2))

    R_action = np.block([[Rx],
                         [Rdx]])
    s_function_Robot_Arm = R_action @ actions  # Assume the actions have a proportional effect on
    the observations
    return s_function_Robot_Arm
```

A.5 State Updates

A.5.1 Solver

```
import numpy as np
import sympy as sym
from numpy import r_
from scipy.integrate import solve_ivp
from sympy import Matrix
from os.path import dirname as up
from os import path

from .update_funcs import mu_update, v_update, a_update
from .free_energy import free_energy
from .diff_equations import cartgroups_AI_dstep

def solve(t_end, dt, dx, params, gen_mod, states, states_init, learning_rates, noise_vars, noise,
    symbolic, filename):
    # unpack
    [x, s, a, mu, v] = states
    [x0, s0, a_0, mu_0, v_0] = states_init
    [kmu, kv, ka] = learning_rates
    [interp_noise, noise_w, noise_z] = noise
    [D, s_function] = gen_mod[-2:]

    observation_noise = noise_vars[1]
    noise_vars = Matrix(noise_vars)

    print('calculate free energy expression')
    F = free_energy(gen_mod, mu, v, s)

    print('Belief and action update expressions')
```

```
    dmu = mu_update(F, mu, kmu, D)
    dv = v_update(F, v, kv)
    da = a_update(F, s, s_function, a, ka)

    print('Create dstep function')
    s_noisy_dict = list(zip(s, x + observation_noise))  # List of pairs of sensory observations adding noise
    dmu = dmu.subs(s_noisy_dict)
    dv = dv.subs(s_noisy_dict)
    da = da.subs(s_noisy_dict)

    s0 = s0 + interp_noise(0)[0:len(s)]                    #initial value of the sensory information

    initial_values = r_[x0, mu_0, v_0, a_0]
    solver_states = Matrix([x, mu, v, a])
    dstep = sym.lambdify((solver_states, noise_vars), Matrix([dx, dmu, dv, da]))

    #Determinant calculation
    if symbolic == True:
        step = Matrix([dx, dmu, dv, da])
        M = step.jacobian(solver_states)
        print(M)

    print('Solve differential equations')

    time_span = np.r_[0:t_end:dt]

    solver_emergency_stop_xc.terminal = True
    solver_emergency_stop_dtheta.terminal = True
    sol = solve_ivp(cartgroups_AI_dstep, [0, t_end], initial_values, method='RK45', t_eval=time_span, vectorized=True,
                    rtol=1e-4, atol=1e-7, args=(dstep, interp_noise))
    sol_t = sol.t

    x_index = len(x)
    mu_index = x_index + len(mu)
    v_index = mu_index + len(v)
    a_index = v_index + len(a)
    sol_x = sol.y[0:x_index]
    sol_mu = sol.y[x_index:mu_index]
    sol_v = sol.y[mu_index:v_index]
    sol_a = sol.y[v_index:a_index]
    sol_save = (params, dt, sol_t, sol_x, sol_mu, sol_v, sol_a, noise_w, noise_z)
    print('done')

    data_folder = up(up(__file__)) #The current directory, used to specify the absolute path to save animations
    data_filename = filename + '.npy'
    sol_file = open(data_filename, 'w+')
```

```
    data_filepath = path.join(data_folder, 'data', data_filename)
    np.save(data_filepath, sol_save)
    sol_file.close()

    solution = [sol_t, sol_x, sol_mu, sol_v, sol_a]
    return solution
```

A.5.2 Free Energy

```
import numpy as np
from sympy import flatten
from sympy import Matrix
from numpy.linalg import det
from scipy.linalg import block_diag

# The approximated Free Energy

def free_energy(gen_mod, mu, v, s):
    [f, g, precision_w, precision_z, D_mu] = gen_mod[0:5]

    s_v = Matrix([s, v])
    e_mu = D_mu @ mu - f
    e_y = s_v - g

    e = Matrix([e_mu, e_y])
    precision = block_diag(precision_w, precision_z)

    f_e = 0.5 * e.T @ precision @ e #- Matrix([np.log(det(precision))])
    return f_e
```

A.5.3 Update Functions

```
import numpy as np
from sympy import Matrix

def mu_update(f_e, mu, kmu, D):
    dmu = D @ mu - kmu * (f_e.jacobian(mu).T)
    return dmu


def v_update(f_e, v, kv):
    if len(v) == 0:
        dv = Matrix([])
    else:
        dv = -kv * (f_e.jacobian(v).T)
    return dv


def a_update(f_e, s, s_function, a, ka):
    # s are the observed states, which depend on the actions (enacted forces) a.
```

```
    dfdy = f_e.jacobian(s)
    da = -ka * (s_function.jacobian(a).T @ dfdy.T)
    return da
```

A.5.4 Differential Equation for the solver

```
from numpy import absolute

def cartgroups_AI_dstep(t, states, ddx_AI_function, interp_noise):

    states = states.flatten()
    ddx = ddx_AI_function(states, interp_noise(t))

    return ddx
```