Applied Hierarchical Active Inference

on a skid-steering mobile robot

M.T. Deken

Implementation of an hierarchical active inference controller performing online control on a skid-steering mobile robot in continuous state-space

TU Delft

Applied Hierarchical Active Inference

on a skid-steering mobile robot

by

M.T. Deken

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 27, 2021 at 14:00 PM.

Student number:	4294831	
Project duration:	January 1, 2021 – August 1, 2021	
Thesis committee:	Prof. dr. ir. M. Wisse,	TU Delft, supervisor
	Dr. E. L. Brown,	TU Delft, committee
	Dr. C. Della Santina,	TU Delft, committee

TUDelft

Preface

This work is the end piece of a very interesting journey through the topic of active inference. Besides the application for control, it also taught me a different view on how organisms (and the brain) function, which can put things in a different perspective. I would like to thank Martijn for his supervision during this thesis and the preceding research project. Also a thank you to the whole active inference group for the interesting meetings and insights, in particular Charel for his notebooks on active inference, making it easier to step into the subject. Lastly, special thanks to my parents and girlfriend for their support and patience during my studies.

M.T. Deken
Zuid-Scharwoude, July 2021

Abstract

Active inference is a novel brain theory based on the free energy principle, stating that every organism, in order to stay alive, minimizes a certain free energy. This theory is being translated into robot control, hoping to mimic the capabilities of the brain. Research in this field of robotics is still quite young, and active inference has yet to mature into a reliable control theory. Implementations of active inference are scarce, and hierarchical implementations, containing multiple layers of active inference, are almost non-existent. This work presents the first implementation of hierarchical active inference performing online control on a robot in continuous state-space. The robot in question is a skid-steering mobile robot, on which two different active inference controllers are implemented and compared. The first controller is a non-hierarchical active inference controller giving wheel speed commands to the default PID controller of the robot manufacturer based on desired body velocities. The second controller is a novel implementation of a hierarchical active inference controller, that besides the function of the first controller, also replaces the low-level PID control with active inference, resulting in two-layers of active inference ultimately taking desired body velocities as inputs and translating it into open loop voltage commands for the motors. Both controllers are tuned to give roughly the same step responses in order to get a clearer view on the differences between non-hierarchical and hierarchical active inference. During the tuning process of the hierarchical controller, it became clear that no amount of tuning could get the (causal) states to convergence sufficiently fast. This problem was identified to be caused by the message passing in between the layers of active inference, where the prediction error from below negatively impacts convergence. This impact scales together with the tuning parameters, and therefore is not easily fixed by tuning alone. A potential solution is given in the form a an extra parameter in front of the upwards prediction error, giving the ability the independently tune the impact of this error on the behaviour of the system. This solution is used to fully tune the hierarchical controller, after which the performance is compared to the non-hierarchical controller, using a series of experiments. A total of 8 experiments are done where both controllers are tested performing two driving actions in two situations, namely cornering and pivoting on the ground and with the wheels suspended. The non-hierachical controller, as a baseline, performed satisfactory, but the hierarchical controller does not run smoothly on the ground, potentially struggling with the stick-slip of the wheel-ground interactions. From this it seems that using active inference for the low-level, as compared to something like PID, is less suitable. However, that conclusion is too premature for this work as the available encoder feedback frequency for the active inference is way lower than what the MCU-level PID has access to, making the comparison less fair. The main contribution of this work lies in the insight in the implementation structure of hierachical active inference for continuous state-space control, which can be further built upon, using the lessons learned during implementation.

List of symbols

$\mathcal{F}$	free energy function/formula
μ	beliefs (non-hierarchical)
μ_x	beliefs on the hidden states
μ_v	beliefs on the causal states
y	observations or measurements
a	actions
ε_μ	prediction error on the beliefs (non-hierarchical)
ε_x	prediction error on the beliefs about the hidden state
ε_v	prediction error on the beliefs about the causal states
ε_y	prediction error on the observations
$\dot{\mu}$	update rule on the beliefs (non-hierarchical)
$\dot{\mu_x}$	update rule on the beliefs about the hidden states
$\dot{\mu_v}$	update rule on the beliefs about the causal states
$\dot{a}$	update rule on the action
$g(\mu)$	generative function mapping beliefs to observations (non-hierarchical)
$g_y(\mu_x)$	generative function mapping hidden states to observations
$g_v(\mu_x)$	generative function mapping hidden states to causal states
$f(\mu_x)$	generative function encoding how the hidden states change over time
Π	precision matrix (= Σ^{-1})
Σ	covariance matrix
η	prior or user command (also denoted as μ_{des})
$\mathcal{D}$	shift operator, shifting values of a vector 1 spot upwards
$\frac{\delta y}{\delta a}$	forward model
c	coefficient for the prediction error part in the update rule for the causal states

Contents

1 Introduction

1.1. Active Inference

Active Inference is a novel brain theory based on the free energy principle. The free energy principle entails the notion that every living organism or non-living system, in order to maintain a certain non-equilibrium steady-state (staying alive), will try to minimize their free energy. The free energy can be described as a mismatch between the system's internal model of the world and the actual real world process. One way to decrease this mismatch is to adjust the internal model to better match the real world process, by means of perception. The other way is to get the real world process to get closer to the internal model. This last part is done by means of performing actions on the world, hence the 'active' in active inference.

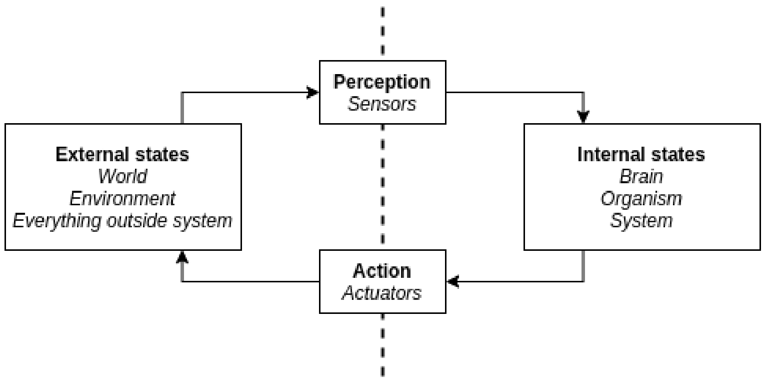


Figure 1.1: Schematic overview of how the interaction takes place between internal and external states

This theory was first introduced in neuroscience by Karl Friston [5], and to this day a lot of research still resides on the neuroscientific side, supported largely by Friston himself. These last years, however, a transition is underway where this theory is being translated into control theory for robotics. Its motivation being that if this is truly how the brain works, translating it to robotics potentially means getting closer to 'true artificial intelligence', and therefore better robot control. Because if we want to get robots to behave more like humans, what better way to do that is to try to mimic how the brain works? Our brain has the incredible ability to handle uncertainties and complex problems. These uncertainties can stem from inexperience, but also uncertainties in perception, i.e. an obstructed view. Robots suffer from similar problems, i.e. learning of a model, and the limited information a sensor provides, but are still far away from handling them like humans can.

Nonetheless, great efforts are being made to get closer to this goal. These efforts started out mostly as offline simulations [3, 10, 18], but recently more and more implementations are on real world robots doing online closed-loop control [14, 16, 2]. The added value of these real world implementations, apart from it being the end goal, is that much can be learned from early implementations, even though some things might not yet be fully thought in theory. It sheds light on implementation problems early and ultimately give solutions that directly work in the real world. An example of this is the addition of a temporal parameter by Baioumy et al [2], and this work will also serve as an example.

These efforts have now created a state in research where, although a lot of aspects still need improvement, the implementation structure of an active inference controller is quite clear when talking about simple systems, with a small amount of inputs and outputs, residing close to the action where the actions still have a recognizable relationship with the goal. But for more complex systems, like our brain, this single-layer 'barebones' structure (as described in section 2.1.2) is not enough.

This is where hierarchy comes into play. For quite some time there has already been the notion that the brain works in an hierarchical manner [4]. Karl Friston also gives an hierarchical model of the brain [6] explaining how hierarchical active inference works in theory. Only a few efforts have been made to translate this theory and use hierarchical active inference in practice, where some only include perception [19, 9], others focus on machine learning [12], and the ones for robot control are only done in simulations [21].

The purpose for this work is to fill the implementations gap of hierarchical active inference applied to a real-world robot doing online closed-loop control. This work not only shows how hierarchical active inference is implemented on a skid-steering mobile robot, but also shows how it expanded from an earlier non-hierarchical implementation that the author also worked on (see Chapter 4). This non-hierarchical implementation serves as a baseline to compare against in terms of structure, tuning and performance. This baseline is chosen to keep the focus on the differences between non-hierarchical and hierarchical active inference, contrary to stacking it up against other types of control, as raw performance is not the main goal. To still get an indication of performance, a series of experiments are conducted at the end that compare the two controllers while cornering and pivoting, both on the ground, and with the wheels suspended.

1.2. Research goal

The detailed research goal is given below:

Implement a two-level hierarchical active inference velocity controller on a skid-steering mobile robot, and compare its structure, tuning and performance with that of its non-hierarchical equivalent

The focus of this works predominantly lies with the practical implementation of hierarchical active inference, and less with the theory behind it. This is also where most of the labour went, in programming an active inference controller in C++ within the ROS framework, and getting it to work on the research platform (Jackal robot).

Figure 1.2: The Clearpath Jackal robot in the lab (for scale), tethered to a computer running an active inference node

An additional result of this thesis is a codebase[1] of an hierarchical active inference controller that has the flexibility to adapt to other applications as well. Even when the controller produced in this

[1] https://gitlab.tudelft.nl/active-inference-drive-control/active-inference-drive-control/-/tree/hierarchical_aic

thesis might still be quite 'barebones', with adequate simplifications and assumptions (as mentioned in section 9.1), it will hopefully also serve as a good starting point for further research on hierarchical active inference utilizing the ROS framework.

In order to kickstart the implementation of active inference on the research platform, a team had been setup of four master students, including the author, with the goal of getting a (non-hierarchical) active inference controller working on the Jackal robot as fast as possible. From this work, that is collabaratively summarized in Chapter 4, every contributor could fork into their own theses. This work expands the collectively built controller to an hierarchical version and then compares it with the original, as shown in Figure 1.3.

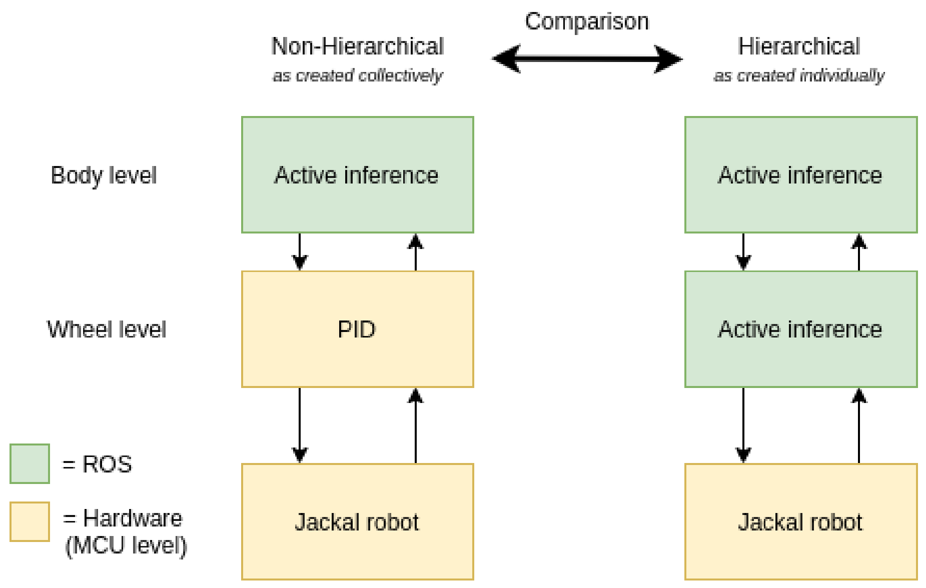


Figure 1.3: Schematic overview of the comparison that is going to be drawn in this thesis

With the non-hierarchical implementation still making use of a low-level controller for the regulation of the wheels speeds, an opportunity has presented itself to replace this low-level controller (PID) with an active inference controller. In this way an hierarchical active inference controller can be created while keeping the inputs the same, namely desired body level velocities (linear and angular velocity). The output differs however, as the hierarchical version outputs open loop wheel voltage commands to the wheels, and the non-hierarchical version outputs wheel speed commands to the low-level PID controller. This makes it not a pure comparison between forms of active inference, as the PID of the manufacturer is also part of the control loop of the non-hierarchical version. This influences mostly the comparison of performance (Chapter 7), and not the comparison in terms of controller structure (Chapter 4 & 5) and tuning (Chapter 6).

1.2.1. Sub-goals

In order to arrive at the research goal of this thesis, it is divided into sub-goals. These are defined as follows.

- Compare the controller structure of non-hierarchical and hierarchical active inference, showing how one expands from the other
- Get a 'working' solution of non-hierarchical and hierarchical active inference running on the Jackal robot
- Tune both controllers and report differences in tuning techniques
- Compare the performance of both controller

Besides the goals for this thesis, some guidelines were followed for the practical implementation (coded solution) as well, where it was desired to stay as close to the theory as possible. The implementation should be modular, and therefore easily expandable to more than two levels of hierarchy. The implementation should also be a good foundation where future improvements, regarding hierarchical active inference, can be built upon.

1.3. Thesis overview

This work will continue with giving some additional background information on active inference in general and the theory behind hierarchical active inference in Chapter 2. In Chapter 3,the research platform is explained to give an idea of the hardware that is being controlled. An overview on how the non-hierarchical controller is implemented as a collective is given in Chapter 4. This is followed, in Chapter 5, by the hierarchical implementation, this work's main contribution in terms of implementation structure. After the structure of both controllers is know, they are tuned, of which the findings are reported in Chapter 6, including a convergence problem together with a potential solution. In Chapter 7, both controllers are compared in terms of performance, having both controllers conduct a series of driving tasks both on the ground and with the wheels suspended. The main conclusions are summarized in Chapter 8, with a following discussion in Chapter 9, containing the simplifications and assumptions made in this work, and a recommendation on interesting further studies.

2 Background information

The purpose of this chapter is to give some background information on how active inference can be implemented in robotics, giving the equations needed and the controller structure. Additionally, the theory behind hierarchical active inference is discussed. The information given in this chapter mostly covers the information needed for implementation rather than the mathematical explanations and derivations behind the equations.

2.1. Active inference for robotics

As mentioned in the introduction, every organism tries to minimize their free energy in order to stay alive. This free energy characterizes the mismatch between the system's internal model of the outside world and the real process of the outside world. The real process is not fully known by the system and is only partially observed through perception, which is why the system can only calculate the free energy by comparing its internal model to what is observed. The system takes all of its decisions based on the minimization of this free energy, including updating its internal states and taking actions.

Taking decisions in this way makes it that a controller based on active inference behaves different from most controllers, where actions are directly, through gain or a model, calculated from the (filtered) observations. Instead of working with states that are derived directly from filtered observations, the system keeps a belief about its inner states that, although indirectly connected to observations, gets updated independently. This can be seen as a form of filtering, making the active inference controller serve as both the controller and the filter. The update of the beliefs, together with the update of the actions the system is taking, are both done in such a way that it minimizes the free energy function.

The behaviour described above is very similar to that of optimal control (i.e. LQR and MPC) in the sense that it minimizes a certain cost function. The key difference, however, is that active inference, optimizes a cost function (free energy) for both perception and action, meaning it acts both as a filter and a controller. Like MPC it also optimizes for the future, by keeping a belief about the transition of the states. The way it does this is by storing information about the higher order derivatives of states, called generalized coordinates of motion. For example, if a state describes the position of an object, the system also stores information about its velocity, acceleration, jerk etcetera. The amount of higher order derivates that are stored by the system is called the embedding order. Friston [5] talked about the embedding order of an active inference system to be a maximum of 6, as after that the information would not be of much value anymore. In this work we will be working with an embedding order of 3, which is chosen quite arbitrarily during the tuning process in Chapter **??**, where the tuning parameters for these higher orders are chosen such that they don't influence the system a lot, because the method used for extracting the higher orders, as described in section 4.4 is quite sub-optimal and diverge greatly for higher orders.

2.1.1. Continuous and discrete state space

Active inference can be applied to both the continuous and discrete state space. When comparing to nature, it can be seen that the brain can take discrete decisions on a high level (planning), for example taking either a left or right turn, while the same brain can handle the continuous control of the muscles to make sure that the discrete decision is fulfilled. This thesis will focus on the continuous state space implementation of (hierarchical) active inference. The structure and equations given in this thesis apply to the continuous state space, and although the principle stays the same, it cannot be used for discrete control, as that introduces other nuances.

2.1.2. Structure overview

The give an overview of the controller structure, in Fig. 2.1 a simplified diagram of the structure of an active inference controller is given. This diagram is a novel contribution, serving as the author's interpretation of a simplified active inference controller, based on earlier implementations of non-hierarchical active inference [16, 2]. This visualization of the structure should also be a reference of where the upcoming equations are going to be in effect.

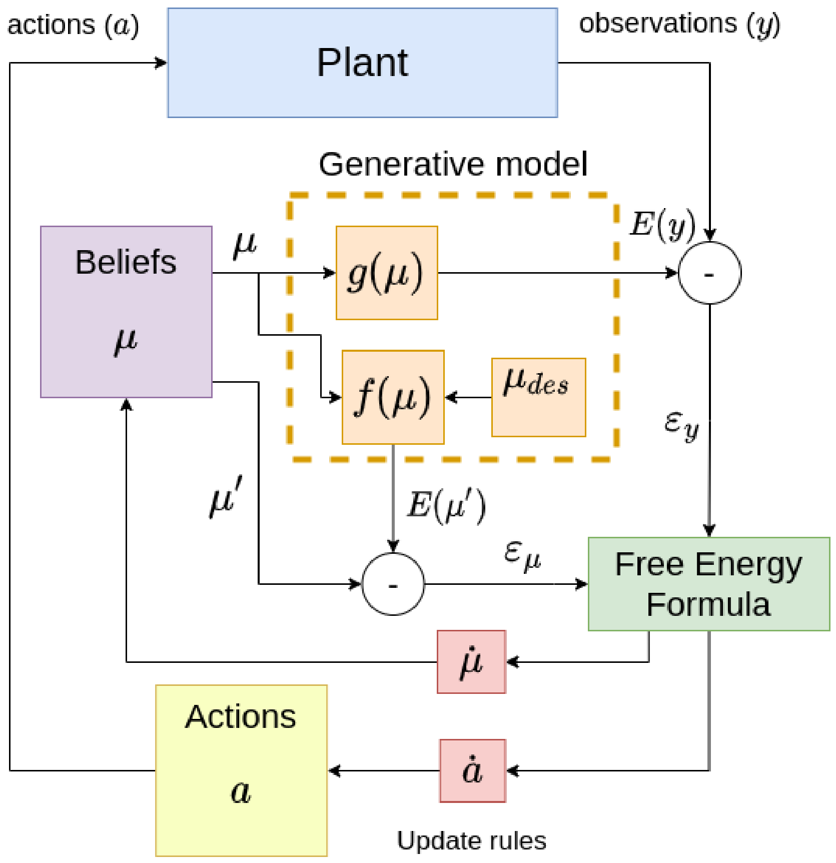


Figure 2.1: Simplified overview of the structure of an active inference controller

Starting from the beliefs about the internal states, by means of a generative model an expectancy $E(y)$ about what will be observed is created, together with an expectancy $E(\mu')$ about how its beliefs transition in time. These expectancies get compared with the incoming observations y and the actual transition in time μ' respectively to create the errors ε_y and ε_μ. These errors are the main ingredients to the free energy formula, and characterize the mismatch between the internal model and the external process as mentioned earlier. Subsequently, from the free energy formula the best update for the beliefs $\dot{\mu}$ and the actions $\dot{a}$ are chosen based on the gradients $\frac{\delta \mathcal{F}}{\delta \mu}$ and $\frac{\delta \mathcal{F}}{\delta a}$ by means of a gradient descent. This gradient basically entails that if for example gradient $\frac{\delta \mathcal{F}}{\delta a}$ turns out positive, meaning increasing the a will increase the free energy, it chooses to decrease a as that will minimize the free energy. The chosen actions get pushed to the plant (in this work a skid-steering mobile robot) and then the cycle repeats, creating a continuous feedback loop.

2.1.3. The free energy formula

A simplified version of the free energy formula can be seen in eq. 2.1, as used in other implementations in robotics [16, 2]. A full derivation of the formula will not be given, instead a notion on where it stems from. The free energy is based on Bayesian principles, where the marginal likelihood (model evidence) needs to be maximized in order to give an accurate prediction on the occurrence of states. This means giving a probability of the current states occurring in every single situation, caused by any source. This gives an intractable solution, resulting in the simplification where the free energy is a best estimate of the divergence (difference) between its internal model evidence and the real model evidence.

$$\mathcal{F} = \frac{1}{2}\sum_{i=0}^{p}[\varepsilon_y^{(i)T}\Pi_{y^{(i)}}\varepsilon_y^{(i)} + \varepsilon_\mu^{(i)T}\Pi_{\mu^{(i)}}\varepsilon_\mu^{(i)}] \tag{2.1}$$

Where p is the embedding order of the system, meaning that the total free energy consist of its ’0th order’ free energy plus the free energy of every embedding order (higher derivative). The free energy includes the errors ε_y and ε_μ, which are squared and then multiplied by their respective precision matrices Π_y and Π_μ. The errors are created based on the difference between the expectation based on inner beliefs and what can be observed. These errors are given below:

$$\begin{aligned} \varepsilon_y &= y - E(y) = y - g(\mu) \\ \varepsilon_y' &= y' - g(\mu') \\ &\vdots \\ \varepsilon_y^{(p)} &= y^{(p)} - g(\mu^{(p)}) \end{aligned} \tag{2.2}$$

$$\begin{aligned} \varepsilon_\mu &= \mu' - E(\mu') = \mu' - f(\mu) \\ \varepsilon_\mu' &= \mu'' + f(\mu') \\ &\vdots \\ \varepsilon_\mu^{(p)} &= \mu^{(p)} + f(\mu^{(p)}) \end{aligned} \tag{2.3}$$

The expectancies $E(y)$ and $E(\mu')$ are created with the generative functions $g(\mu)$ and $f(\mu)$ respectively. $g(\mu)$ denotes the (often kinematic) mapping between the beliefs and observations, and $f(\mu)$ is a dynamic model that encodes how it beliefs the states change over time. $f(\mu)$ predicts what μ' will be based on μ (or μ''' based on μ'' etc.), then this prediction gets compared with the current μ'. The generative functions together form the system’s generative model.

2.1.4. The Generative model

The generative model is the system’s internal model of the generative process of the plant, like in Fig. 2.1. The generative process of an LTI-system as defined by Grimbergen [7] is of the form:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + w(t) \\ y(t) &= Cx(t) + z(t) \end{aligned} \tag{2.4}$$

with the generalized form:

$$\begin{aligned} \mathcal{D}\tilde{x} &= \tilde{A}\tilde{x} + \tilde{B}\tilde{u} + \tilde{w} \\ \tilde{y} &= \tilde{C}\tilde{x} + \tilde{z} \end{aligned} \tag{2.5}$$

Where w and z are process and measurement noise, respectively. The $\mathcal{D}$ operator is a shift operator that shifts every value of a vector one spot upwards, and is given as a matrix full of zeroes, with ones on the superdiagonal:

$$\mathcal{D} = \begin{bmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix} \otimes I_n \tag{2.6}$$

The matrix shown has a size corresponding with the embedding order ($p \times p$) and I_n corresponds to the amount of different states n ($n \times n$ matrix). The generalized form of A, B and C correspond to:

$$\begin{aligned} \tilde{A} &= I_p \otimes A \\ \tilde{B} &= I_p \otimes B \\ \tilde{C} &= I_p \otimes C \end{aligned} \tag{2.7}$$

Assuming the noises w and z to be Gaussian and a mean of zero, the eq. 2.5 can be written as:

$$\begin{aligned} \mathcal{D}\tilde{x} - \tilde{A}\tilde{x} - \tilde{B}\tilde{u} &= 0 \\ \tilde{y} - \tilde{C}\tilde{x} &= 0 \end{aligned} \tag{2.8}$$

This formulation transcribes to the process and measurement errors used in active inference as follows:

$$\begin{bmatrix} \varepsilon_\mu \\ \varepsilon_y \end{bmatrix} = \begin{bmatrix} \mathcal{D}\tilde{\mu} - \tilde{A}\tilde{\mu} - \eta \\ \tilde{y} - \tilde{C}\tilde{\mu} \end{bmatrix} \tag{2.9}$$

Where Bu is replaced by η, denoting a prior expectation of the states, which can be compared to giving the system a reference signal like in traditional control. This means that for a LTI-system, the generative functions $f(\mu)$ and $g(\mu)$, as seen in eq. 4.7 and 2.2 can be defined as:

$$\begin{aligned} f(\mu) &= A\mu - \eta \\ g(\mu) &= C\mu \end{aligned} \tag{2.10}$$

The generative function $f(\mu)$ can be seen to both have a dynamic model A and its preference for the prior η encoded in its definition. This works for a simple LTI-system, but the problem however is when systems become more complex and this A is not that easily known by the system, and in addition can no longer be written as a linear matrix multiplication. In that case it is not yet fully defined yet how and if such nonlinear models should be implemented into active inference. Because on the one hand active inference is explained to have an internal model, but on the other hand that model should ideally not be bound to a structure based on any known equations (like the laws of physics), as for example when trying to rotate an object the brain does not first calculate its inertia based on the second law of motion.

This brings us to the implementations of Pezzato [16] and Baioumy [2], where they successfully implemented a version where the dynamics of the system are omissed and the $f(\mu)$ function only consists of the preference to go to the prior by means of attractor dynamics. Pezzato [16] defined the $f(\mu)$ function as follows:

$$f(\mu) = \mu - \eta \tag{2.11}$$

Instead of using the prior η directly, it is encoded in the $f(\mu)$ function is a trajectory towards η. Baiyoumy et al. [2] expanded this function towards to contain a temporal parameter τ, which is basically a gain parameter, that lets you have more control over this trajectory towards η (for example a faster convergence).

$$f(\mu) = (\mu - \eta)\tau^{-1} \tag{2.12}$$

By means of simplicity, this work uses the $f(\mu)$ function as defined in eq. 2.12, meaning no dynamic model of a skid-steering mobile robot is needed or defined in this work. The $g(\mu)$ function that is used is unchanged from eq. 2.10, making this function the only applications specific one of the two. It is linear mapping that maps beliefs to observations, which for example in our specific application of a mobile robot means a steady-state kinematic mapping of the beliefs of the body-level angular and linear speed towards expected measured wheels speeds. The detailed description of how these functions are defined for our specific application (skid-steering mobile robot) can be found in Chapter 4.

2.1.5. The update rules

The update rules make sure that every update of the beliefs or actions are chosen in such a way that it minimizes the free energy. It does this with the help of the respective gradients $\frac{\delta\mathcal{F}}{\delta\mu}$ and $\frac{\delta\mathcal{F}}{\delta a}$ and with learning parameters κ_μ and κ_a that form a type of gain on those gradients.

$$\dot{\tilde{\mu}} = \mathcal{D}\tilde{\mu} - \kappa_\mu \frac{\delta\mathcal{F}}{\delta\tilde{\mu}} = \mathcal{D}\tilde{\mu} - \kappa_\mu \left(\frac{\delta\tilde{\varepsilon}_\mu}{\delta\tilde{\mu}}^T \frac{\delta\mathcal{F}}{\delta\tilde{\varepsilon}_\mu} + \frac{\delta\tilde{\varepsilon}_y}{\delta\tilde{\mu}}^T \frac{\delta\mathcal{F}}{\delta\tilde{\varepsilon}_y} \right) \tag{2.13}$$

$$\dot{a} = -\kappa_a \frac{\delta\mathcal{F}}{\delta a} = -\kappa_a \frac{\delta\tilde{\varepsilon}_y}{\delta a}^T \frac{\delta\mathcal{F}}{\delta\tilde{\varepsilon}_y} \tag{2.14}$$

where the gradients of eq. 2.13 are easily calculated from derivatives. In eq. 2.14, however, the gradient $\frac{\delta\tilde{\varepsilon}_y}{\delta a}$ is where the impact of the actions on the observations are needed to be known. This is where the forward model comes in.

2.1.6. Forward model

The forward model describes the system's knowledge of how the actions relate to the observations. This relations needs to be known to be able to predict how updating an action influences the free energy. The gradient $\frac{\delta\tilde{\varepsilon}_y}{\delta a}$ needs to be known, and in order to calculate this the forward model (or gradient) $\frac{\delta y}{\delta a}$ is needed. In simple systems this equates to a linear matrix transformation as shown below:

$$y = H \cdot a \tag{2.15}$$

$$\frac{\delta y}{\delta a} = H \tag{2.16}$$

For the more complex systems the same discussion as in subsection 2.1.4 arises where it is possible to incorporate a complex (forward) model, but that kind of superseeds the goals for active inference. Instead [16] showed that the system also works with just knowing the signs within this forward model. This can result in a forward model filled with only 1, -1 or 0, indicating whether changing the action has a positive, negative or no effect on the observations. In this work's implementation of hierarchy this last type of forward model is used, as can be seen in Chapter 5. For the non-hierarchical implementation a linear transformation is partly used in the forward model, as can be seen in Chapter 4.

2.1.7. Precision Matrices

The precision matrices Π_μ and Π_y as defined in eq. 2.1 are the parameters that in theory says something about how reliable the error signals ε_μ and ε_y are. This includes information about the noise, where more noise equals a lower precision. The precision matrices can be calculated from variance as follows:

$$\Pi_\mu = \Sigma_\mu^{-1} = \begin{bmatrix} (\frac{1}{\sigma_1})^2 & 0 & 0 \\ 0 & (\frac{1}{\sigma_2})^2 & 0 \\ 0 & 0 & (\frac{1}{\sigma_3})^2 \end{bmatrix} \tag{2.17}$$

$$\Pi_y = \Sigma_y^{-1} = \begin{bmatrix} (\frac{1}{\sigma_1})^2 & 0 & 0 \\ 0 & (\frac{1}{\sigma_2})^2 & 0 \\ 0 & 0 & (\frac{1}{\sigma_3})^2 \end{bmatrix} \tag{2.18}$$

The above situation is specifically for a situation where the noise is assumed independent, which is why only the diagonal is filled. σ_1, σ_2 and σ_3 are, depending on the precision matrix, the variances of the signals μ_1, μ_2 and μ_3 or y_1, y_2 and y_3. As can be seen, for this example the amount of belief states are assumed to be 3 and measurement states as well, in reality these amounts can of course differ and depend on the application. Because generalised coordinates of motion are being used, besides having different variances in between the different states (i.e. μ_1, μ_2, μ_3 etc.), the variance also differs between the higher orders (i.e. μ_1, μ_1', μ_1'' etc.). The generalized precision matrices, for an example of 2 embedding orders can be seen below:

$$\tilde{\Pi}_\mu = \begin{bmatrix} \Pi_\mu & 0 & 0 \\ 0 & \Pi_\mu' & 0 \\ 0 & 0 & \Pi_\mu'' \end{bmatrix} \tag{2.19}$$

$$\tilde{\Pi}_y = \begin{bmatrix} \Pi_y & 0 & 0 \\ 0 & \Pi_y' & 0 \\ 0 & 0 & \Pi_y'' \end{bmatrix} \tag{2.20}$$

Choosing the values of the precision matrices is not so straightforward. For Π_y the variance of the measurements y can be calculated and used as these are quite consistent and is independent of the controller, this will give quite a good approximation. For Π_μ, however, it is less clear on what values should be taken as the variance also depends on the controller parameters (including its own precision). Not knowing exactly how to setup these precision matrices is not that big of a problem, this is because of the way these precision matrices are used. In practice these matrices serve as weight parameters that determines the ratio of influence between not only ε_μ and ε_μ, but also between different states and additionally between the higher orders of these states. This makes it for example possible with the precision matrices to make the system rely more on its beliefs, favor particular measurements, and by giving more weight to a higher order include more damping. This results in the precision matrices, although brought as something to be determined from the process, actually being an extra tuning parameter to the active inference controller. This is the reason why in this work the precision matrices seem to be quite arbitrarily chosen as to just make the system 'work'.

2.1.8. Generalised coordinates and generalization

Active inference makes use of generalised coordinates of motion for the beliefs of all its states. This means also keeping track of higher derivatives of states. Keeping beliefs this way means that it does not only have a belief on its current state, but also a belief on its future states, because if you know the higher derivatives you can predict future states, for example by means of Taylor expansion. This means that keeping those higher order beliefs, in theory, should also minimize free energy on future states, making the minimization more effective.

There is a problem though, for the calculation of $\tilde{\varepsilon}_y$ a generalized form of the observations y is needed. However, these higher order measurements are not available to the system, and therefore needs to be established. This is done by generalization. The most simple form of this is backwards differentiation, this takes measurements from the past and the differences between them to establish the higher order derivatives that are needed for the generalized form. This is only an approximation however, because the measurements from the past only gives you derivatives from the past, which does not have to equate the current derivatives. How this is implemented is further explained in Chapter 4. Other techniques can also be used to try to predict the current derivatives, an example could be the use of a predictive coding network (PredNet).

$$\tilde{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_1' \\ \mu_2' \\ \mu_1'' \\ \mu_2'' \end{bmatrix} \tag{2.21}$$

The above equation shows an example of how a vector of generalized beliefs looks like for a system of 2 states and an embedding order of 2. An updated version of the structure overview in Fig. 2.1 can be seen below in Fig. with the incorporation of generalized coordinates and generalization.

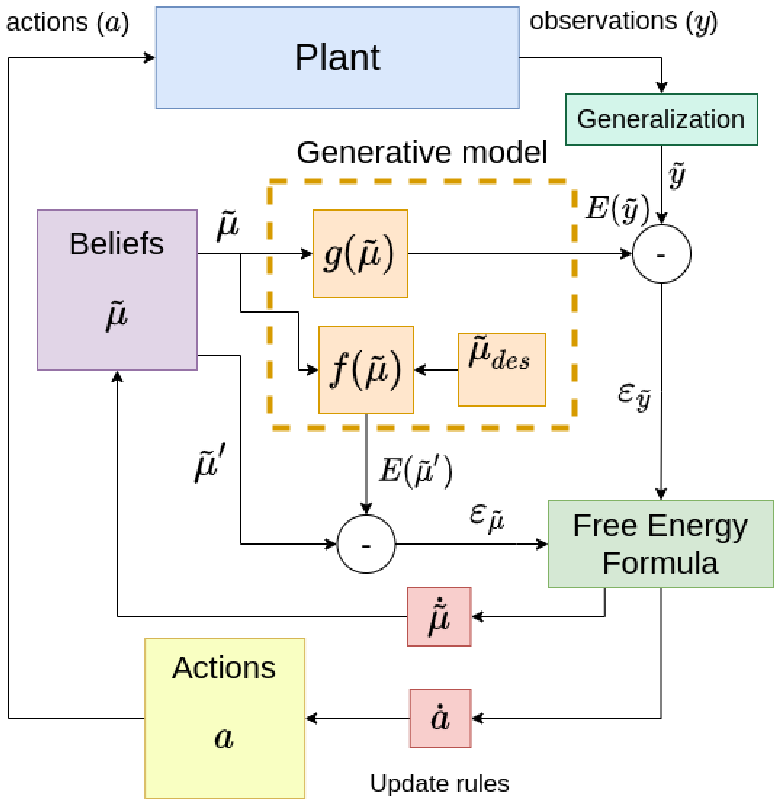


Figure 2.2: updated overview of the controller structure incorporating generalised coordinates and generalization

2.2. Hierarchical active inference

This section gives an overview of what principles gets added when you expand active inference to a hierarchical system, and the interpretation of the author on how it changes the controller structure.

Hierarchical active inference could be seen as active inference inference on top of active inference, meaning that the output of one active inference layer is the input of another. However, it is more nuanced than just connecting outputs and inputs. There are two main aspects that get changed in hierarchical active inference. The first one is the addition of causal states μ_v, and the second one lies in the message passing or information exchange between levels of hierarchy.

Only the lowest level in hierarchy is taking actions on the environment, this means that for levels higher than the lowest level, the actions a and its update rule get replaced by the causal state μ_v with its own update rule. The causal states are the system's beliefs about what the causes are to the system's current hidden states μ_x (formerly known as just internal state μ), for example a causal state of a force that corresponds to a hidden state of an acceleration. The hidden state is what was previously called beliefs or internal states in general, denoted as μ in Fig. 2.2. In non-hierachical implementations the internal states can also be called hidden states, but the usage of this name becomes more prominent when using hierarchy, because of the addition of the causal state. This causal state is the link between levels of hierarchy and serves as a prior on the active inference layer below and enters at the f function of its generative model, shown as μ_{des} in Fig. 2.2.

A schematic illustration of a two-level hierarchical system is given in Fig. 2.3, in which the interaction between the hidden and causal states are visualized. The η is the most high level prior, which in robotics basically entails the desired state the user gives to the active inference controller. With more levels of hierarchy, this user command becomes more abstract as well, for example instead of 'move arm x amount' it could handle 'grab object x'. The lowest levels of hierarchy that commands actions, and therefore reside closer to the hardware, are typically also the levels where the observations y flow into. It is however still possible for higher levels to gather observations, and thus the implementation mentioned in this work can handle both.

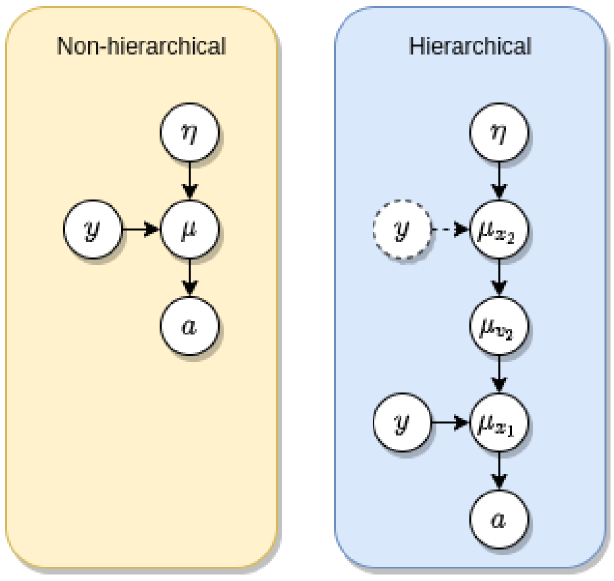


Figure 2.3: Schematic illustration of state interaction in non-hierarchical vs. hierarchical (two-level)

Hierarchical active inference is often depicted with the use of circles connected with arrows, like in Fig. 2.3. This notation stems from neuroscience, and is often used to describe the system in a high level. Papers discussing hierarchical active inference often talk about it on the higher level, not showing equations [17], and the papers that do [15], do not show them fully worked out and do not show them in relation to a controller structure. In Chapter 5, this work visualizes the changes in controller structure when going from non-hierarchical to hierarchical, and shows the corresponding changes in equations, including the values used for the application specific parameters (i.e. generative model, forward model and mappings).

3 Experimental Setup

This chapter gives an overview of the robot that is being used as the research platform for the implementation of Active Inference. In addition it will explain some of the choices that were made for the implementation tools and will explain why the comparison is drawn between hierarchical and non-hierarchical active inference, and not compared with another type of controller.

3.1. Jackal robot

The Jackal robot is a four-wheeled mobile research platform from Clearpath. It maneuvers around with the use of skid-steering, comparable to how an armored tank steers. The Jackal robot is equipped with wheel encoders and an inertial measurement unit (IMU), making it possible to get feedback on wheel speeds, angular velocity and linear accelerations. There is only one encoder measurement per side, as the wheels are mechanically linked from front to back.

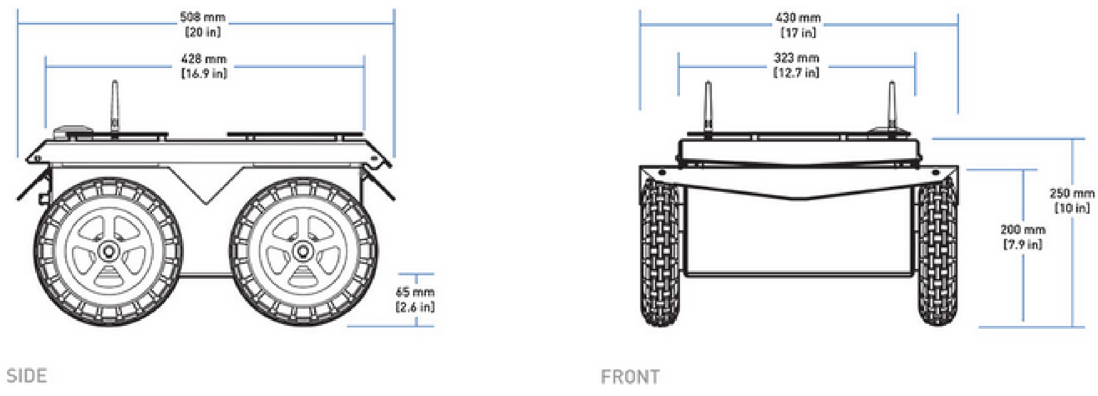


Figure 3.1: Jackal side and front view[1]

The available measurement frequencies for the wheel encoders and the IMU are 20Hz and 70Hz respectively. One thing to note is that especially the wheel encoder feedback frequency is rather low, making it more difficult to do low level wheel speed control. The feedback is also after transmission, making the readings less accurate. We do suffer from these low frequencies, as with this work's implementation of hierarchical active inference, the low-level speed control of the wheels get replaced. That control was previously done by means of PID on the micro-controller unit (MCU) to which the user does not have access. Now, with active inference, the implementation is bound to the 20Hz measurements that are available within ROS.

Full sizing and specifications can be found on Clearpath's website[1]

[1]`https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/`

There are three control modes for the Jackal robot. The first one is through kinematic commands, where a desired linear and angular velocity is given to the robot, then the stock controller will turn those into wheel commands . The second control mode is via wheel velocity commands, where the user gives desired wheel speeds, for which the low-level PID will do the regulating. The third one is giving open loop voltage commands to the motor driver. Control mode 2 is where the non-hierarchical controller is built on top of. Control mode 3 is where the hierarchical controller is built upon.

The stock controller for control mode 1 is not used in this work, but it is interesting to note that this controller uses a differential drive controller, which calculates wheel commands based on a forward kinematic model. This forward kinematic model is similar to the model that is used in this work for the matrices of the generative and forward model within active inference. This forward kinematic model is called *extended differential drive*, and is the most common kinematic model used in skid-steer mobile robots. It relates body-level velocities to wheel velocities in the following way:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ -\frac{1}{\hat{b}} & \frac{1}{\hat{b}} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{3.1}$$

where r is the radius of the wheels, and $\hat{b}$ is the virtual width of the wheelbase as if it had two wheels instead of four. For the Jackal robot these values are $r = 0.098\,m$ and virtual width $\hat{b} = 0.5621\,m.$ The virtual width was determined empirically using two different wheel speeds as input, and measuring the corresponding angular velocity with the IMU.

3.2. ROS and C++

The robot has an on-board computer that runs a version of the Robot Operating System (ROS). ROS is based on a system of nodes where each node runs its code independently and subscribes and/or publishes values to a topic.

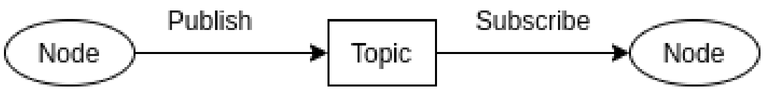


Figure 3.2: ROS structure, where topics are illustrated with rectangles and nodes are displayed as ellipses

ROS nodes can be programmed with either C++ or Python. Although Python makes for better readability, benefitting knowledge transfer, the author's choice was to go with C++, based on prior experience and the Jackal's stock ROS nodes also being in C++. Furthermore, the decision has been made to not implement the controller as a *ros_controller*, a framework for control within ROS. The control is done with independent nodes, where the interaction with the user and the hardware are manually programmed. One reason for this is the lack of experience with *ros_control* of the implementation team that built the non-hierarchical version, which highlights an additional benefit as a potential next student working with the same code does not first have to familiarize themselves with *ros_control*. Another reason is that building it from scratch, means that the controller is not tied to traditional control structure and can have a structure that expands horizontally and vertically in hierarchy, like the brain. The full overview of the implementation structure within ROS for the hierarchical implementation can be found in Appendix B.

3.3. Tuning experiments

In order for the comparison of the non-hierarchical and hierarchical controller to be fair, they both have to be tuned in the same way. As they both have different structures, it is not possible to tune to roughly the same tuning parameter values. To keep the comparison somewhat fair, the tuning of both controllers is done using roughly the same tuning strategy as explained in Chapter 6. Tuning is done until the step response on the wheel measurements settles fast enough (about <0.5s) to the expected values for both controllers. For the non-hierarchical controller, it was possible to do some initial tuning on the Jackal simulation in Gazebo, however this simulation was not able to take voltage command inputs for the wheels (command mode 3), making it impossible for the hierarchical version to be tuned in this way. Eventually all tuning, including non-hierarchical, was done with experiments on the real robot.

These experiments were done with the wheels suspended, with the Jackal placed on a box, making it so that the robot would not crash into things if the controller rendered unstable. After the on-box tuning (setup showed in Fig. 3.3), the robot is ready to be placed on the ground, these results are discussed in Chapter 7.

3.4. Comparison experiments

When the controllers are done being tuned, they are ready to be compared to eachother. The reason these controllers are being compared, as opposed to comparing with a traditional controller, is to keep the focus on the differences between non-hierarchical and hierarchical active inference, as opposed to raw controller performance. These controllers both achieve the same task, making the wheels turn based on a desired linear and angular body velocity. The non-hierarchical controller, however, is built on top of a MCU-level PID controller doing the low-level wheel speed control with access to much higher frequency encoder data, as opposed to the low frequency (20Hz) the low-level active inference layer of the hierarchical controller has access to in ROS. This makes a performance comparison between the two less fair, which is why the main focus does not lie in comparing performance, but more with a comparison of what it takes to get both controllers to behave in the same way. As mentioned earlier, the baseline (non-hierarchical) is built by a group of students, including the author.

Eventually both controllers are compared with a total of 8 experiments, 4 experiments per controller. This includes driving in a circle (cornering), and turning in place (pivoting) for both on the ground and with its wheels suspended in the air.

	on box	on ground
pivoting	exp. 1	exp. 3
circle driving	exp. 2	exp. 4

Table 3.1: the different experiments done for both controllers

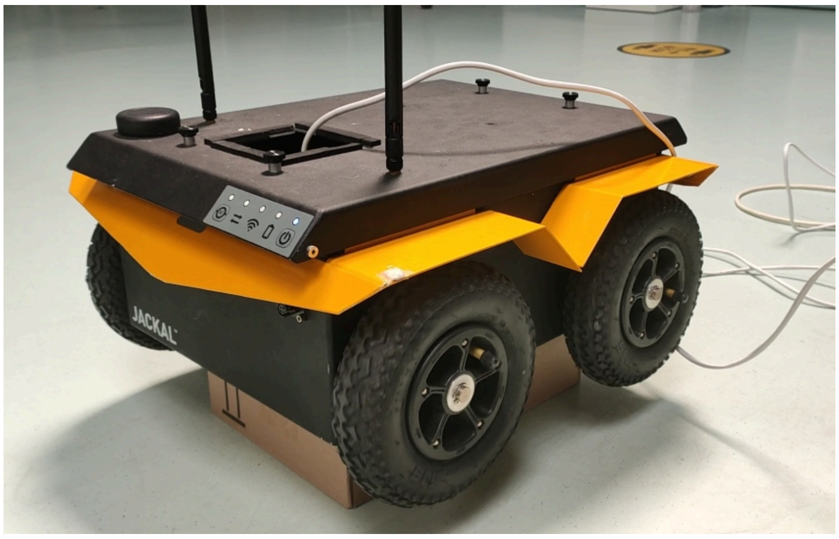


Figure 3.3: A photo of the Jackal robot for the the on-box experiments, where the wheels are suspended in the air

4

Non-Hierarchical Implementation

This chapter was written in collaboration with 3 other Masters students as a documentation of the collective effort to get a first implementation of Active Inference running on the Jackal robot.

This chapter explains the active inference velocity controller that was created to track the velocity of the Jackal UGV. In section 4.1 an overview of the controller is given, section 4.2 describes the chosen dynamical model while section 4.3 explains the free energy formulation. Lastly section 4.4 explains the generalization node of the controller.

4.1. Overview of the active inference velocity controller

The controller for the Jackal robot needs to send and receive information to and from the plant. This communication is handled by ROS, which is inherently present on the Jackal robot. There are many topics available, but for the controller only a few are needed. An overview is shown in Figure 4.1. Sensory input from the wheel speeds are published on the topic /joint_states, while the speed of the center of mass μ is published on /imu/data, containing $\ddot{x}$, $\ddot{y}$ and $\dot{\theta}$. The sensory data are read by a generalization node that calculates the derivatives of the wheel speeds. These derivatives are then published on an intermediate topic: /sensor_generalized. These generalized measurements, combined with the desired velocity of the center of mass that is published on /jackal_aic/set_point, are the actual input for the AIC. The AIC calculates the action required to bring down the Free Energy and publishes drive commands on /cmd_drive when using the robot and /jackal_left_wheel/command and /jackal_right_wheel/command when using Gazebo. These topics contain the velocities that the left- and right wheel of the Jackal should have: ω_L and ω_R. Ideally an immediate connection to the actuators would be used, but that is not yet implemented. Currently an internal lower-level controller transforms the drive commands to actuator voltages.

4.2. Choice of dynamic model

For the control of a skid-steering mobile robot, often no dynamical model is used [11, 1, 13, 22, 20]. Instead, only a kinematic mapping between the body velocity and the wheel velocities is often used. This gives a steady-state solution of the body velocity resulting from the wheel velocities. In this way, using inverse kinematics, the desired body velocity input by the user (linear and angular) can be converted by the controller to give corresponding wheel speed inputs to the wheels. This method however, does not contain any dynamics, and therefore cannot adjust well for any disturbances, like unknown slippage.

Dynamic modelling takes the forces and accelerations into account. This will generally lead to an increase in control performance because the vehicle is modeled on a deeper level. However, this type of modeling only makes sense when you know the torques that are applied to each wheel, otherwise it is best to stick to a kinematic model. To keep things simple, the dynamic model is often only used in 2D (top-down view of the vehicle) where a simple 2D chassis model is implemented. The forces acting on the chassis, that determine its motion, are the driving forces and friction forces. The driving forces can be derived from the motor torques, whereas the friction forces on the chassis come from the wheel-ground interaction. For this last part an abundance of wheel-ground interaction models (or friction

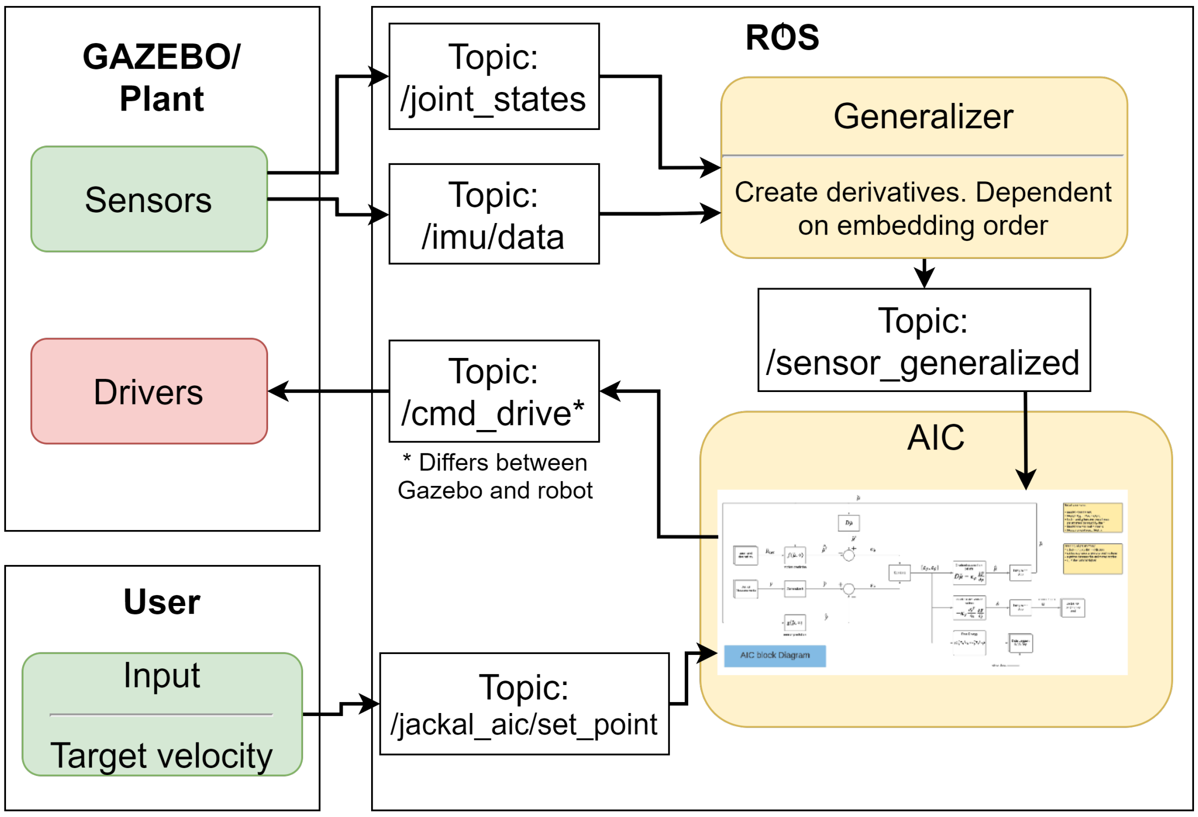


Figure 4.1: An overview of all components involved and through what ROS topics they communicate. A full-page view of the block diagram depicted within the AIC block can be found in the Appendix A.

models) are used in literature, varying widely in complexity. But taking even the simplest friction model, the resulting overall dynamic model will still be non-linear. For the implementation of active inference, we are looking for a simple solution, as we are focusing on the implementation in the hardware, instead of focusing on complex models. Therefore, we chose to opt out of a traditional dynamic model, and use attractor dynamics instead. This technique is already used in a few active inference implementations [16, 2], where we will use the version with the time constant [2]. These attractor dynamics will encode a preference of the states to be moving towards a prior (desired state), stated as follows:

$$f(\mu) = (\mu_d - \mu)\tau^{-1} \tag{4.1}$$

where μ are the beliefs (states in active inference), μ_d is the prior belief (desired state) and τ is a time constant.
While not using any dynamic model from literature, we still need a mapping from the states to observations (body velocities to wheel velocities) in order to compare with measured values. For this we opted for the extended differential drive model (as used by [11]), because this is the most simple and most used kinematic mapping available, while still giving sufficient performance:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = r\alpha \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ -\frac{1}{\hat{b}} & \frac{1}{\hat{b}} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{4.2}$$

$$r\alpha \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \\ -\frac{1}{\hat{b}} & \frac{1}{\hat{b}} \end{bmatrix} = G \tag{4.3}$$

where α is a slip parameter and $\hat{b}$ is the virtual width of the vehicle (as if it had two wheels). These are both trained empirically. For the $g(\mu)$ of the generative model, we need the mapping from the states to observations. And as we have access to the gyroscope, encoder and accelerometer measurements, it will look as follows:

$$\begin{bmatrix} \dot{\theta} \\ \omega_L \\ \omega_R \\ \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ G_{11}^{-1} & G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1} & G_{22}^{-1} & G_{23}^{-1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{4.4}$$

where G^{-1} is the pseudo-inverse of the kinematic mapping mentioned above.

4.3. Free energy formulation

This section will focus on the definition of the free energy formula for the active inference velocity controller of the Jackal robot. In the previous paragraph the generative model of the sensory data and of the state dynamics are defined. Equation 5.1 displays the general free energy formula as it was defined by Pezzato et al.[16]

$$F = \frac{1}{2} \sum_{i=0}^{n_d - 1} [\varepsilon_y^{(i)(T)} \Sigma_{y^{(i)}}^{-1} \varepsilon_y^{(i)} + \varepsilon_\mu^{(i)(T)} \Sigma_{\mu^{(i)}}^{-1} \varepsilon_\mu^{(i)}] \tag{4.5}$$

Based on the models defined earlier the error terms, ε_y and ε_μ, can be computed. These can be found in equations 4.6 and 4.7. Furthermore, to find the first generalized order these questions are differentiated once to obtain equations 4.8 and 4.9. Note that in this example only the first derivative is shown. However, in the actual application the derivatives can go up to the 5th order.

$$\varepsilon_y = y - g(\mu) = y - R\mu = \begin{bmatrix} \dot{\theta} \\ \omega_L \\ \omega_R \\ \ddot{x} \\ \ddot{y} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{r\alpha} & 0 & \frac{-\hat{b}}{2r\alpha} \\ \frac{1}{r\alpha} & 0 & \frac{\hat{b}}{2r\alpha} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{4.6}$$

$$\varepsilon_\mu = \mu' - f(\mu) \tag{4.7}$$

$$\begin{aligned} \varepsilon_y &= y - R\mu \\ \varepsilon_y' &= y' - R\mu' \\ &\vdots \\ \varepsilon_y^{(5)} &= y^{(5)} - R\mu^{(5)} \end{aligned} \tag{4.8}$$

$$\begin{aligned} \varepsilon_\mu &= \mu' - f(\mu) = \mu' - (\mu_d - \mu)\tau^{-1} \\ \varepsilon_\mu' &= \mu'' + \mu'\tau^{-1} \\ &\vdots \\ \varepsilon_\mu^{(5)} &= \mu^{(6)} + \mu^{(5)}\tau^{-1} \end{aligned} \tag{4.9}$$

For readability, equations 4.8 and 4.9 can be written in matrix form (equation 4.10). Furthermore, the same can be done for the precision matrices, which is done in equation 4.11. Note that in this case $\tilde{\varepsilon}$ and $\tilde{\Pi}$ consist of all generalized orders.

$$\tilde{\varepsilon} = \begin{bmatrix} \tilde{\varepsilon}_y \\ \tilde{\varepsilon}_\mu \end{bmatrix} \tag{4.10}$$

$$\tilde{\Pi} = \begin{bmatrix} \tilde{\Pi}_y & 0 \\ 0 & \tilde{\Pi}_\mu \end{bmatrix} \tag{4.11}$$

Combining equations 5.1, 4.10 and 4.11 equation 4.12 can be found. This is the free energy formula for the active inference velocity controller for a jackal robot.

$$F = \frac{1}{2}\tilde{\varepsilon}^T \tilde{\Pi} \tilde{\varepsilon} \tag{4.12}$$

From the free energy formula the belief update can be found, the belief update is defined in equation 4.13. In equation 4.14 this has been written out for 2 generalized orders.

$$\dot{\tilde{\mu}} = D\tilde{\mu} - k_\mu \left(\frac{\delta\tilde{\varepsilon}_\mu}{\delta\tilde{\mu}} \frac{\delta F}{\delta\tilde{\varepsilon}_\mu} + \frac{\delta\tilde{\varepsilon}_y}{\delta\tilde{\mu}}^T \frac{\delta F}{\delta\tilde{\varepsilon}_y}\right) \tag{4.13}$$

$$\begin{bmatrix} \dot{\mu} \\ \dot{\mu}' \\ \dot{\mu}'' \end{bmatrix} = \begin{bmatrix} 0 & I & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mu \\ \mu' \\ \mu'' \end{bmatrix} - k_\mu \left(\begin{bmatrix} \frac{I}{\tau} & 0 & 0 \\ I & \frac{I}{\tau} & 0 \\ 0 & I & \frac{I}{\tau} \end{bmatrix} \begin{bmatrix} \Pi_\mu \varepsilon_\mu \\ \Pi_{\mu'} \varepsilon_{\mu'} \\ \Pi_{\mu''} \varepsilon_{\mu''} \end{bmatrix} - \begin{bmatrix} R^T & 0 & 0 \\ 0 & R^T & 0 \\ 0 & 0 & R^T \end{bmatrix} \begin{bmatrix} \Pi_y \varepsilon_y \\ \Pi_{y'} \varepsilon_{y'} \\ \Pi_{y''} \varepsilon_{y''} \end{bmatrix} \right) \tag{4.14}$$

Lastly the action update is defined in equation 4.15. This has been written out for 2 generalized orders of motion in equation 4.17. Note that the term $\frac{\delta\tilde{\varepsilon}_o}{\delta a}$ is replaced by the forward model (F) as defined in equation 4.16.

$$\dot{a} = -\kappa_a \frac{\delta\tilde{\varepsilon}_y}{\delta a} \frac{\delta F}{\delta\tilde{\varepsilon}_y} \tag{4.15}$$

$$F = \begin{bmatrix} \frac{-r\alpha}{\hat{b}} & 1 & 0 & 0 & 0 \\ \frac{r\alpha}{\hat{b}} & 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.16}$$

$$\dot{a} = -\kappa_a \begin{bmatrix} F & 0 & 0 \\ 0 & F & 0 \\ 0 & 0 & F \end{bmatrix} \begin{bmatrix} \Pi_y \varepsilon_y \\ \Pi_{y'} \varepsilon_{y'} \\ \Pi_{y''} \varepsilon_{y''} \end{bmatrix} \tag{4.17}$$

4.4. Generalization

Active Inference also requires the measurements to be generalized. The states of μ are generalized easily by just taking multiple derivatives. However this is not as straight forward for generalizing measurements.

There is only a limited amount of derivatives real sensors can measure. Think of a position sensor, a velocity sensor and an acceleration sensor. These link the derivatives directly through measurements of each other. In this case this would mean that we can only measure up to an embedding order of two directly. But what if some of these type of sensors are not available or what if the highest state is acceleration and we need derivatives of that? In that case we need to synthesize the higher order derivatives from the available measurements.

4.4.1. Finite differences

The following part is based off the Master's thesis of I.L. Hijne [8]. A robot receives measurement samples in discrete time. In that case we can use a simple differentiating technique by using finite differences. It is possible to extract higher order derivatives from a Taylor expansion. The Taylor expansion computes signal y_{k+j} from y_k and a weighted sum of y_k's higher order derivatives and is defined as:

$$y_{k+j} = \sum_{i=0}^{i_{max}} \frac{(jh)^i}{i!} y_k^{(i)} \tag{4.18}$$

where (i) denotes the i-th order derivative of y_k and h is an interval step that is sufficiently small. Note that i_{max} goes to infinity and j is used for looking j steps forward/backwards and note that $0^0 = 1$.

The goal here is to extract every order of derivatives up to a defined i_{max}. So for the 1st order we can write the Taylor expansion as:

$$\begin{aligned} y_{k-1} &= y_k - hy_k^{(1)} + \mathcal{O}(h^2) \\ y_k^{(1)} &= \frac{1}{h}(y_k - y_{k-1}) + \mathcal{O}(h) \end{aligned} \tag{4.19}$$

Here, $\mathcal{O}(h^n)$ denotes the order of magnitude of the error terms. Similarly, we can do this approximation for the second derivative by eliminating the terms until we only end with the second order terms on the right side.

When

$$\begin{aligned} y_k &= y_k \\ y_{k-1} &= y_k - hy_k^{(1)} + \frac{1}{2}h^2 y_k^{(2)} + \mathcal{O}(h^3) \\ y_{k-2} &= y_k - 2hy_k^{(1)} + 2h^2 y_k^{(2)} + \mathcal{O}(h^3) \end{aligned}$$

are combined linearly to eliminate all but the second order terms we find.

$$\frac{1}{2}y_k - 1y_{k-1} + \frac{1}{2}y_{k-2} = \frac{1}{2}h^2 y_k^{(2)} + \mathcal{O}(h^3) \tag{4.20}$$

This can then be solved for $y_k^{(2)}$:

$$y_k^{(2)} = \frac{1}{h^2}(y_{k-2} - 2y_{k-1} + y_k) + \mathcal{O}(h) \tag{4.21}$$

and has the coefficients $c = [1, -2, 1]$.

The error term still scales linearly with the step h. This error term will be the same for higher order derivatives as well. However this error term can be reduced by taking more past samples into consideration. For example if we took 4 past samples for calculating $y_k^{(2)}$, then the error term scales quadratic with the interval step h: $\mathcal{O}(h^2)$.

Now the problem turned into finding the appropriate coefficients of the linear combinations of Taylor series. I.L. Hijne [8] tabulated these coefficient up to the 5th order derivative for an accuracy of $O(h)$. The coefficients for zero to higher order derivatives can be written in a 6×6 matrix and is shown in eq (4.22). This solution has error terms that scale by $O(h)$.

4.4.2. Matrix forms

In Active Inference the generalized coordinates of the sensory input is for embedding order p:

$$\tilde{\mathbf{y}} = [y_k^{(0)}, y_k^{(1)}, ..., y_k^{(p)}]$$

We can approximate $\tilde{\mathbf{y}}$ by using backwards differentiation of numerous previous samples of $y_k^{(0)}$. This can be written as:

$$\tilde{\mathbf{y}} = E\check{\mathbf{y}}, \qquad \check{\mathbf{y}} = [y_{k-s}, y_{k-s+1}, ..., y_k]$$

In this equation s is the amount of samples backwards. For $p = 5$ and $s = 5$ and an error term of $O(h)$ the system of equations looks like eq (4.22).

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{1}{h} \\ 0 & 0 & 0 & \frac{1}{h^2} & -\frac{2}{h^2} & \frac{1}{h^2} \\ 0 & 0 & -\frac{1}{h^3} & \frac{3}{h^3} & -\frac{3}{h^3} & \frac{1}{h^3} \\ 0 & \frac{1}{h^4} & -\frac{4}{h^4} & \frac{6}{h^4} & -\frac{4}{h^4} & \frac{1}{h^4} \\ -\frac{1}{h^5} & \frac{5}{h^5} & -\frac{10}{h^5} & \frac{10}{h^5} & -\frac{5}{h^5} & \frac{1}{h^5} \end{bmatrix} \begin{bmatrix} y_{k-5} \\ y_{k-4} \\ y_{k-3} \\ y_{k-2} \\ y_{k-1} \\ y_k \end{bmatrix} \tag{4.22}$$

Interestingly, the absolute values of the coefficients follow the same pattern as in Pascal's triangle. This matrix does solve the problem and we get the generalized coordinates of the sensory input with a constant h. However for an implementation of this it is required to keep track of old data samples, which causes the information to have overlap. This equation can be rewritten to only contain data of the sample we are interested in and the previous sample plus its derivatives. A special vector $\check{\mathbf{y}}$ can be constructed that contains the current sample and the previous sample with higher order derivatives of the previous sample.

$$\tilde{\mathbf{y}} = Q\check{\mathbf{y}}, \qquad \check{\mathbf{y}} = [\mathbf{y}_k, \mathbf{y}_{k-1}^\top]^\top$$

$$\begin{bmatrix} y_k^{(0)} \\ y_k^{(1)} \\ y_k^{(2)} \\ y_k^{(3)} \\ y_k^{(4)} \\ y_k^{(5)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{h} & -\frac{1}{h} & 0 & 0 & 0 & 0 \\ \frac{1}{h^2} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 & 0 \\ \frac{1}{h^3} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 & 0 \\ \frac{1}{h^4} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} & 0 \\ \frac{1}{h^5} & -\frac{1}{h^5} & -\frac{1}{h^4} & -\frac{1}{h^3} & -\frac{1}{h^2} & -\frac{1}{h} \end{bmatrix} \begin{bmatrix} y_k \\ y_{k-1}^{(0)} \\ y_{k-1}^{(1)} \\ y_{k-1}^{(2)} \\ y_{k-1}^{(3)} \\ y_{k-1}^{(4)} \end{bmatrix} \tag{4.23}$$

This form is initialized with the higher order derivatives at zero.

4.4.3. Implementation Summary

The form eq (4.23) is used in the Jackal robot. The sensor measurements are synchronized so that h is the same for all measurements during a time step. Their derivatives are computed based on eq (4.23). However instead of making one large matrix that matches the size of $\check{\mathbf{y}}$, it is more efficient to split this $\check{\mathbf{y}}$ into several single vectors of a measurement $\check{y}_{k,i} = [y_{k,i}, y_{k-1,i}^{(0)}, y_{k-1,i}^{(1)}, y_{k-1,i}^{(2)}, y_{k-1,i}^{(3)}, y_{k-1,i}^{(4)}]$, where each $\check{y}_{k,i}$ is a unique sensor measurement denoted by i. Then use these smaller vectors and the 6×6 matrix Q to calculate its derivatives as:

$$\tilde{y}_{k,i} = Q\check{y}_{k,i} \qquad \check{y}_{k,i} = [y_{k,i}, y_{k-1,i}^{(0)}, y_{k-1,i}^{(1)}, y_{k-1,i}^{(2)}, y_{k-1,i}^{(3)}, y_{k-1,i}^{(4)}]$$

This way Q only has to be calculated once and can be reused for every vector of $\check{y}_{k,i}$.

5 Hierarchical Implementation

This chapter gives an overview of the changes in structure and equations when expanding towards hierarchical active inference, and presents it in a 'building block' fashion. This way of visualizing hierarchical active inference is novel, and hopefully this modular way of representing active inference layers can help build hierarchical implementations in the future. After this visualization, the full structure is given of the hierarchical active inference controller as implemented in the Jackal robot, together with a general overview of the states and observations of the system, and a description of the forward and generative models used.

5.1. Changes in structure

Below on the left, a different representation is given of the structure depicted in Fig. 2.1. This representation makes it easier to show how the structure changes for the hierarchical layers. The change indicated below is to see how one layer of hierarchical active inference (all but the lowest level) differs from the single layer implementation for non-hierarchical active inference. How the structure changes for the lowest level can be seen in Fig. 5.3, and to see how these layers are stacked, a full overview can be seen in Fig. 5.4

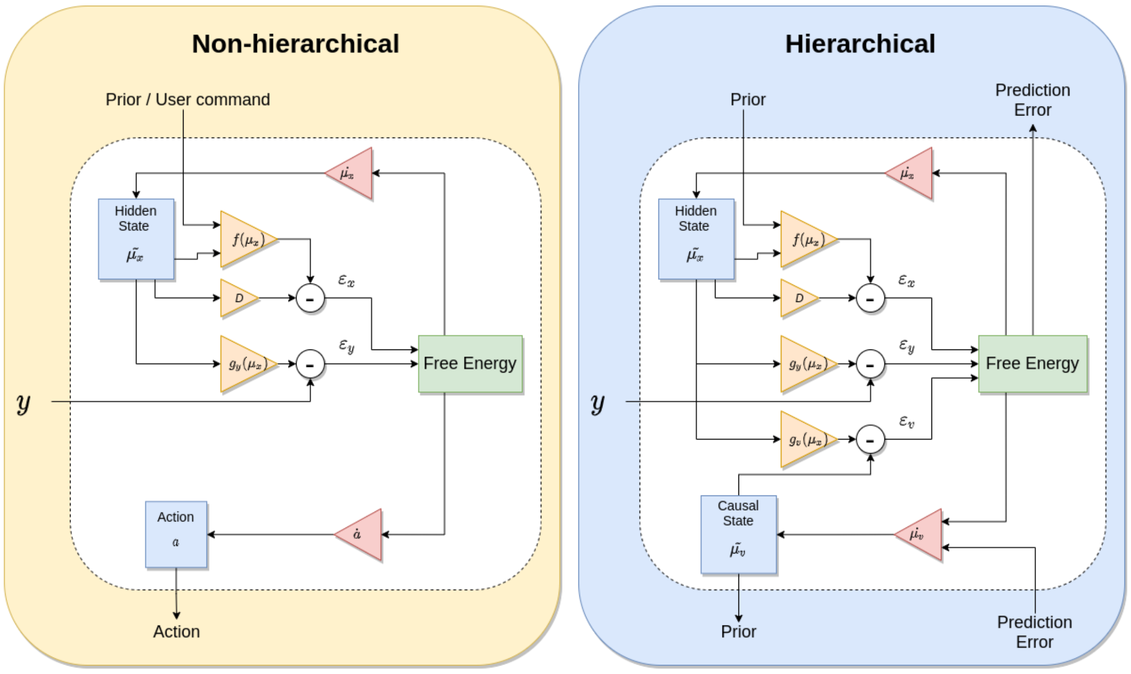


Figure 5.1: Differences in structure between non-hierarchical and hierarchical for higher levels (all but the lowest)

The biggest change takes place in the levels that don't take action but pass down a causal state as prior to the level below, in practice this is every level that has an inference layer below it, or in other words, every level above the lowest level. As can be seen in Fig. 5.1, the action gets replaced by a causal state μ_v. The update rule for this causal state incorporates the free energy of the current level and the prediction error of the level below that got passed upwards. Like the prediction error it gets from below, the layer also needs to pass its prediction error upwards, to be used in an inference layer above. An extra error ε_v gets added to the free energy formula, denoting the error on the causal states, which is calculated by a difference between the causal states and the expectancy of those causal states based on the hidden states. This expectancy is calculated with $g_v(\mu_x)$, which entails a steady-state mapping between μ_x and μ_v. Because this mapping looks a lot like the mapping between μ_x and y, it was given a similar name. The mapping between between μ_x and y, formerly known as just $g(\mu)$, now becomes $g_y(\mu_x)$. The addition of $g_v(\mu_x)$ is novel, as in theory the causal state μ_v gets compared to a prior η to create the error ε_v. The source of this prior about the causal state is not exactly defined in literature [15], but as we only give a manual prior to the hidden state of the top level, it seemed the most logical to calculate the prior for μ_v from the hidden states μ_x of the level it is in, such that $g_v(\mu_x)$ becomes a mapping that indicates what the prior expectancy of the causal states are, given the hidden states. As the hidden states of the top level are influenced by the manually set prior through $f(\mu_x)$, the causal states are therfore also indirectly influenced by the prior, slowed by the update rule $\dot{\mu_x}$, and transformed by $g_v(\mu_x)$.

The observations y shown in Fig. 5.1 is needed for the non-hierarchical controller to function, as it is only one layer, but because the hierarchical controller exists of multiple layers, not every layer necessarily has observations flowing into its active inference loop. Most higher layers of active inference, while still possible, will probably not have access to observations, in which case that part can be omitted from the structure as follows:

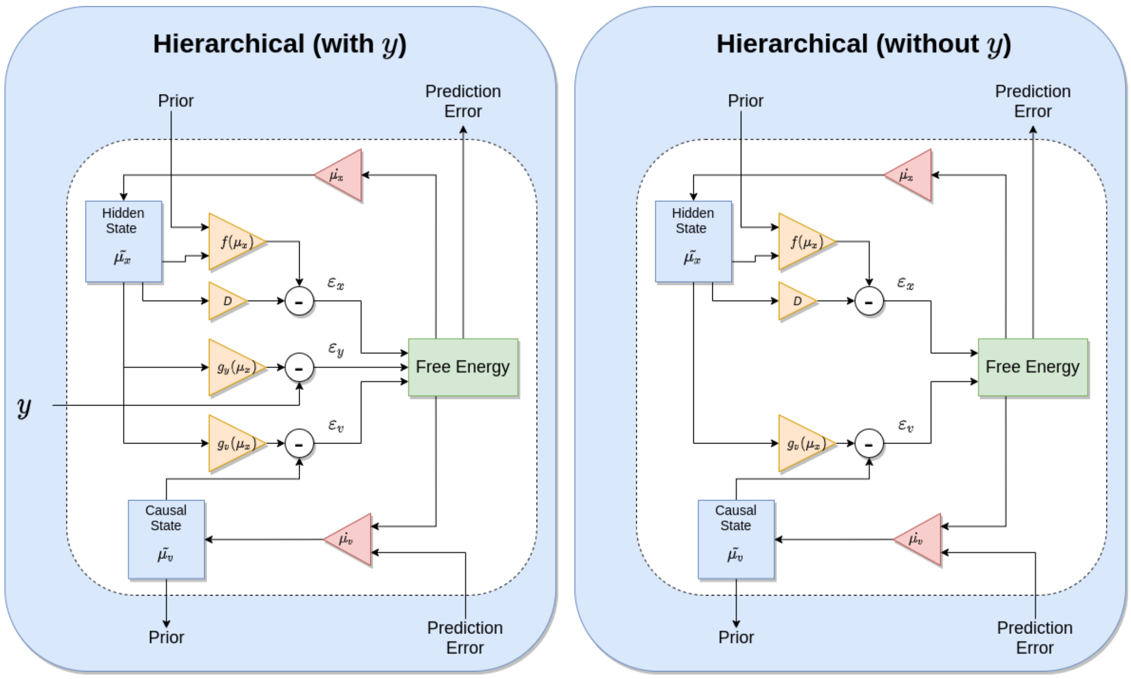


Figure 5.2: Differences in structure between non-hierarchical and hierarchical for the lowest levels

For the lowest layers of active inference not much changes. The only thing that gets added is a prediction error to be passed upwards towards the active inference layers above, as can be seen in Fig. 5.3

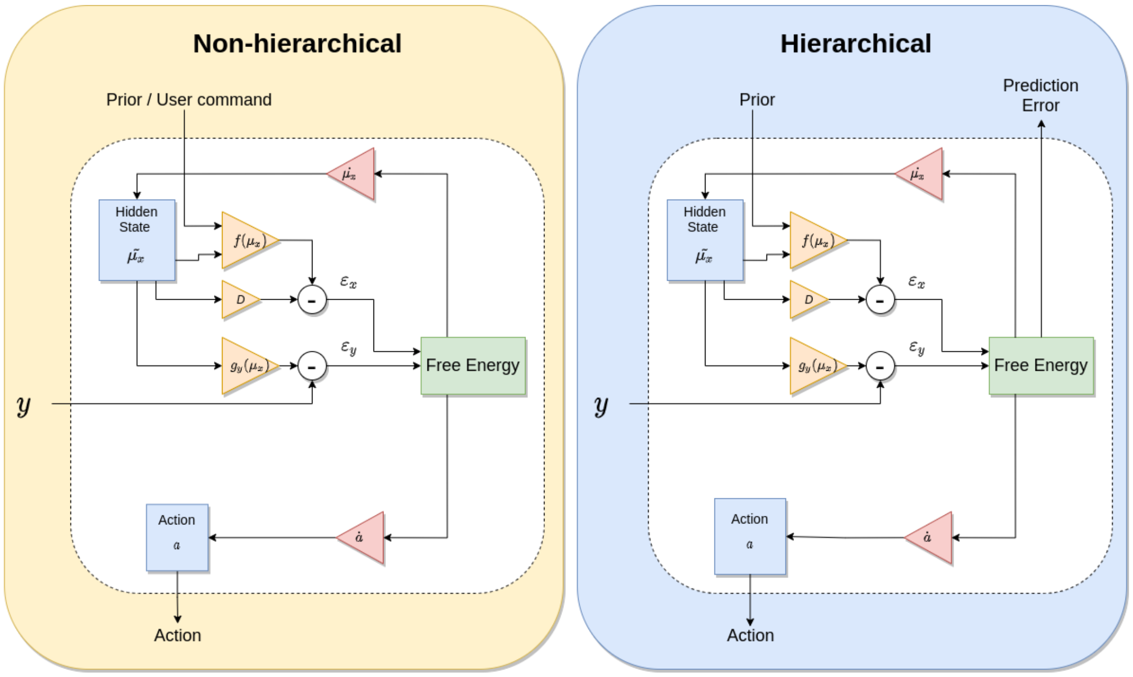


Figure 5.3: Differences in structure between non-hierarchical and hierarchical for the lowest levels

5.2. Changes in equations

The free energy as denoted in eq. 2.1 changes for the higher layers , with causal states, to the following:

$$\mathcal{F} = \frac{1}{2}\sum_{i=0}^{p}[\varepsilon_y^{(i)T}\Pi_{y^{(i)}}\varepsilon_y^{(i)} + \varepsilon_x^{(i)T}\Pi_{x^{(i)}}\varepsilon_x^{(i)} + \varepsilon_v^{(i)T}\Pi_{v^{(i)}}\varepsilon_v^{(i)}] \tag{5.1}$$

The squared error term with ε_v gets added to the equation, and the error formerly known as ε_μ gets his name changed to ε_x. This name change is because both ε_x and ε_v handle beliefs μ, and could also be written as ε_{μ_x} and ε_{μ_v}, but the shorter version was chosen. The addition of error term ε_v, besides being used in the update rule for the causal states μ_v, also changes the behaviour of the update rule for μ_x, such that hidden states are chosen that correspond better to the causal state. ε_v is defined below:

$$\varepsilon_v = \mu_v - g_v(\mu_x) \tag{5.2}$$

$g_v(\mu_x)$ is a mapping of μ_x to the causal state μ_x, and encodes what the system expect the causal states to be with the current hidden states.

The precision matrix Π_v is very similar to Π_x, in a way that it is not quite sure how to determine its values, as discussed in subsection 2.1.7, because its signal variance is dependent on the controller behaviour, which the precision matrix itself also influences.

With the addition of the causal state, it also needs a way of updating. The update rule $\dot{\mu_v}$ for the causal state is defined below in the non-generalized form:

$$\dot{\mu_v} = \mathcal{D}\mu_v - \kappa_v(\frac{\delta\mathcal{F}}{\delta\mu_v} - \frac{\delta\mathcal{F}^{(h-1)}}{\delta\mu_v}) \tag{5.3}$$

where the first part of the equation is very similar to the update rule $\dot{\mu}_x$, with its own learning parameter κ_v. The gradient $\frac{\delta\mathcal{F}}{\delta\tilde{\mu_v}}$ makes sure that the causal state gets updated in such a way that the free energy of the same level gets minimized. The added parameter $\frac{\delta\mathcal{F}^{(h-1)}}{\delta\mu_v}$) is the gradient on how the free energy of the level below gets influenced by a change in μ_v. This works out towards a precision weighted prediction error that gets passed from below, with an additional gradient that says how this passed prediction error is changed by changing μ_v, as can be seen below:

$$\dot{\mu_v} = \mathcal{D}\mu_v - \kappa_v(\frac{\delta\mathcal{F}}{\delta\mu_v} - \frac{\delta\varepsilon_x^{h-1}}{\delta\mu_v}\Pi_x\varepsilon_x^{h-1}) \tag{5.4}$$

The reason that the precision weighted prediction error that is gotten from the level below only consist out of ε_x is because the causal state that gets passed down as a prior only influences μ_x, as can be seen in Fig. 5.2.

The worked out version of eq. 2.13 about μ_x changes to have an extra part added that includes the influence of μ_x on the free energy by means of ε_v.

$$\dot{\mu_x} = \mathcal{D}\mu_x - \kappa_x\frac{\delta\mathcal{F}}{\delta\mu_x} = \mathcal{D}\mu_x - \kappa_{\mu_x}(\frac{\delta\varepsilon_x}{\delta\mu_x}^T\frac{\delta\mathcal{F}}{\delta\varepsilon_x} + \frac{\delta\varepsilon_y}{\delta\mu_x}^T\frac{\delta\mathcal{F}}{\delta\varepsilon_y} + \frac{\delta\varepsilon_v}{\delta\mu_x}^T\frac{\delta\mathcal{F}}{\delta\varepsilon_v}) \tag{5.5}$$

This concludes the section about the general changes for a hierarchical system. In the next sections the specific structure for our implementation is shown, accompanied with the chosen application specific parameters (i.e. generative model, forward model and mappings).

5.3. Controller structure

In Fig. 5.4 the two-level hierarchical structure of the active inference controller is shown. These individual blocks and their equations were already shown in section 2.2, but are now stacked on top of eachother.

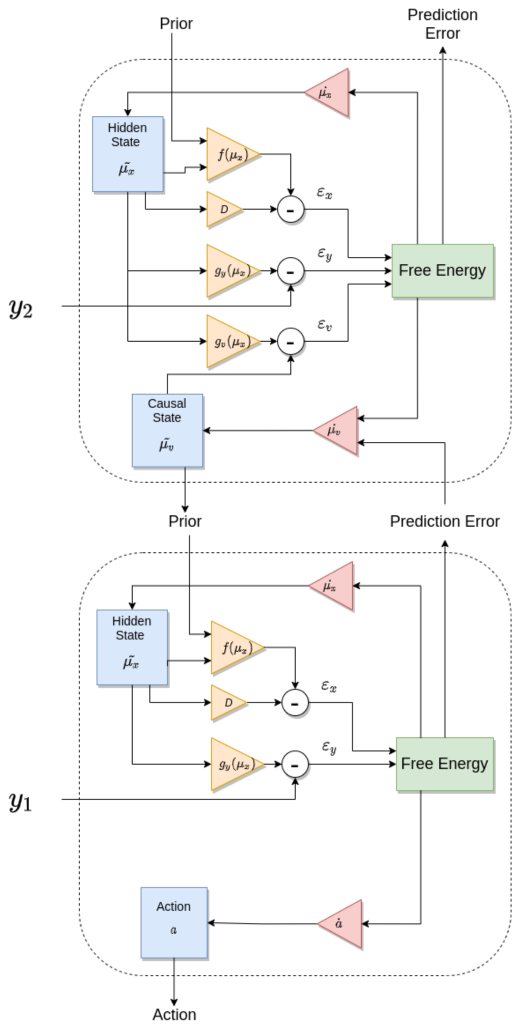


Figure 5.4: Schematic of the hierarchical structure as implemented in the jackal robot

This implementation stacks only two levels of active inference, but the potential of this structure reaches beyond that, as the top level structure can easily be duplicated and stacked on top as many times as the application requires. However, this ’building block’ structure, as depicted in this work, is currently only focused on vertical hierarchical expansion, meaning every inference layer only has a maximum of one inference layer below or on top. This limitation is in place because of the choice to regard the causal state as part of an inference layer (above), while in reality this should actually be a separate entity floating in between the active inference layers with its own update rule containing elements of the free energy of every inference layer that is attached (both as an input or an output). The reason for the structure that has been chosen in this work was partially guided by the fact that, for the implementation in code, it is more practical to have the causal state as part of the active inference layer, as this gives less components overall that has to interact with eachother, meaning less nodes on ROS.

5.4. Overview of states and observations

The hidden states μ_x of the top layer consists of the body level velocities. Subsequently, the prior η on these hidden states is in the form of a commanded angular and linear body velocities from the user. The observations flowing in the top layer only consists of the angular velocity measurements from the IMU. The causal state μ_v that gets updated by this layer contains the left and right wheel speeds that the system thinks are the causes of the body level speeds. These wheel speeds get passed down as a prior. The states for the top layer are given below:

$$\mu_x = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.6}$$

$$\eta = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.7}$$

$$\mu_v = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{5.8}$$

$$y_2 = \begin{bmatrix} \dot{\theta} \end{bmatrix} \tag{5.9}$$

The bottom layer gets the causal state from above as a prior η on the hidden state μ_x, meaning both of them have the same form as eq. 5.12. The observations flowing into this layer are also in this same form as these are wheel speeds as measured by the left- and right-side encoder. The actions are the open loop voltage commands that needs to be pushed to the motor drivers, depicted as PWM-values.

$$\mu_x = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{5.10}$$

$$\eta = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{5.11}$$

$$a = \begin{bmatrix} PWM_L \\ PWM_R \end{bmatrix} \tag{5.12}$$

$$y_1 = \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{5.13}$$

5.5. Generative model

The generative functions that are specific to this application are the different g-functions, $g_y(\mu_x)$ and $g_y(\mu_x)$. The $f(\mu_x)$ function uses the same attractor dynamics as in the non-hierarchical version, and therefore does not change. The generative functions for the top layer are:

$$g_y(\mu_x) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.14}$$

$$g_v(\mu_x) = \begin{bmatrix} \frac{1}{r\alpha} & 0 & \frac{-\hat{b}}{2r\alpha} \\ \frac{1}{r\alpha} & 0 & \frac{\hat{b}}{2r\alpha} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{5.15}$$

The mapping for y is quite straightforward as the hidden state is directly measured. For the mapping towards the causal state μ_v however, the same forward kinematic model is used as in eq. 4.6. This represents a stead-state approximation of the wheels speeds corresponding to the body speeds. r , $\hat{b}$ and α are the wheel radius, virtual width and slip parameter (set to 1.0, could be cut from equation) respectively. For the bottom layer only $g_y(\mu_x)$ is used. Because the hidden states are directly measured, this mapping is just an identity matrix:

$$g_y(\mu_x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{5.16}$$

5.6. Forward model

The forward model $\frac{\delta y}{\delta a}$ is used in the update rule for the action, and therefore only is present in the bottom layer. The forward model relates how a change in open loop voltage commands gives a change in measured wheel speeds. These two are correlated, but next to the voltage, the change in wheel speeds is also dependent on the load on the wheels and the motor parameters. This load is not measured, and therefore a very accurate forward model could not be given, instead only a model based on sign change [16] is used. The only behaviour encoded in this model is the knowledge that if you increase the voltage on the motor, it can be expected that the wheel speed will also increase, assuming the load stays the same. This results again in an identity matrix like in eq. 5.16:

$$\frac{\delta y_1}{\delta a} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.17}$$

For the top layer, there is no forward model for the actions, as there are no actions to be taken. However, the update rule for the causal state μ_v, as depicted in eq. 5.4, consists of the gradient $\frac{\delta \mathcal{F}}{\delta \mu_v}$. This gradient indicates the influence of a change of μ_v on the free energy, and because there are also observations flowing into this level, the gradient $\frac{\delta \mathcal{F}}{\delta \mu_v}$ also partly consists out of the gradient $\frac{\delta \varepsilon_y}{\delta \mu_v}$, indicating the what the change of causal state μ_v has for influence on the observational error ε_y. To know this, a sort of forward model of the causal state towards the observations $\frac{\delta y_2}{\delta \mu_v}$ needs to be known. How this model is exactly defined is not exactly sure, as this is not straightforward, as the feedback loop also runs through an active inference layer before it reaches the plant. As can be imagined, this loop gets more complicated when more layers gets added. For this reason, in this work this gradient is omitted, and therefore assuming the causal state μ_v does not influence the observations y_2. If it is desired to still want to incorporate this, an approximation based on sign change can be given. This encodes the expectation that an increase in left wheel speed will decrease angular body velocity, and an increase in right wheel speed will increase angular body velocity, as depicted below:

$$\frac{\delta y_2}{\delta \mu_v} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.18}$$

That concludes the implementation structure of the hierarchical active inference controller. The system is set up to use the stacking of building blocks on top of one another, but the way these building blocks are defined for this application, still only makes it able to stack vertically, where only one block can be stacked on top of another. To be able to stack multiple blocks on one block, to spread out horizontally in hierarchy, the causal state should behave as a separate block with its own update loop directly connected to one or more active inference nodes above and below. In this work however, the causal state and its update loop is joined together with the active inference loop that resides above it, shown as part of the building block, this is partly done because the hierarchical system was only viewed as purely vertical, and at the end of the work became clear another solution is needed to account for horizontal expansion as well. The decision to keep it inside the active inference loop, was also driven by the the notion that the causal states could be seen as a replacement of the actions for the higher levels. Not having a separate loop for the causal state, also means less nodes (ROS) to implement, but this is only true for the simpler systems, like the one discussed in this work.

6 Tuning

This chapter indicates the tuning technique that was used to tune both the non-hierarchical and hierarchical active inference controller, while reporting on the issues that were encountered during the tuning process. The purpose of tuning, besides getting the robot to work in a satisfactory manner, is to get a roughly similar step response for both controllers, making their comparison more fair.

6.1. Tuning strategy

6.1.1. General

Because there is no simulation of the Jackal robot available for the driving mode (open loop voltage commands) that the hierarchical controller uses, there was no other option than to tune the system in real time. For the safety of its own and surrounding equipment, and to save time with resetting the position, the robot is tuned while residing on top of a box with its wheels suspended in the air. How the behaviour changes when it is put on the ground can be seen in Chapter 7. The tunable parameters for each layer of an active inference system are:

- κ_x, learning rate on the hidden states
- κ_a, learning rate on the action (if bottom)
- κ_v, learning rate on the causal states (if top)
- τ, the temporal parameter inside $f(\mu_x)$
- Π_x, Π_y and Π_v, the precision matrices

Increasing the learning rate κ of a parameter gives a faster convergence of that parameter. Lowering the temporal parameter τ will increase the convergence of μ_x towards the prior η, resulting in a faster step response, but giving a bigger bias towards η, which makes the hidden state react less to changes in observations.

The variances inside the precision matrices should be chosen in such a way that it decreases the noise in the free energy, which in practice means putting a very high variance for every embedding order higher than zero, resulting in almost no influence of these higher orders on the system. Depending on your generalization technique, the generalized measurements can explode in magnitude for higher orders, which means when starting out it is safer to set the variances very high, and deal later with how to incorporate these higher orders better into the system.

The initial learning parameters should be chosen quite low (<1.0) and the temporal parameter set to 1.0, to prevent instabilities, from there you should probably have a system that has a low convergence on both the actions and internal states. If there is still instability, decrease the learning parameters furthers, making sure that, at the start of the explained strategies in the next sections, a stable low-converging system is present

6.1.2. Non-hierarchical tuning order

The order of tuning used for the non-hierarchical system is given as:

1. Get the hidden states μ_x to converge faster, by means of increasing the learning parameter κ_x and/or decreasing the temporal parameter τ.

2. Get the actions a to converge faster, making the measurements resemble the hidden state μ_x by means of increasing the learning parameter κ_x and/or decreasing the temporal parameter τ.

6.1.3. Hierarchical tuning order

The order of tuning used for the hierarchical system is given as:

1. Perform the tuning strategy of subsection 6.1.2 on the bottom layer of active inference while disconnected from its layer above, meaning manually giving a prior η. This results in a bottom layer that is capable of converging fast towards a given prior.

2. For the top layer get the hidden states μ_x to converge faster, by means of increasing the learning parameter κ_x and/or decreasing the temporal parameter τ.

3. Lastly get the causal state μ_v to converge faster by increasing the learning parameter κ_v. The reason this parameter is last is because first the independent layers have to converge well separately before it can be reliably connected by the causal state. As can be seen later in this chapter, getting this to converge is not so straightforward.

The overall order can be seen as bottom-up approach regarding the order of layers to tune, and top-down regarding the tuning of the parameters inside those layers. At least for manual tuning, this is the only distinguished way to get a result. Theoretically, for bigger hierarchical systems (>10 layers), the layers still need to be added one by one on top of the bottom layer and tuned accordingly. If this is done manually, this will takes a lot of time, meaning researching self-tuning algorithms become more interesting.

6.2. Tuning results

6.2.1. Low-level tuning graphs

The tuning results are shown for the tuning of the hierarchical system. First, the bottom layer needs to get tuned separately. In order to tune the system and also get a better understanding of the influence of different parameters, from a certain stable point of convergence, the different tuning parameters are altered independently and the results are logged in graphs. The upcoming graphs show the influence of the tuning parameters, on a non-hierarchical system (as the bottom layer gets tuned independently), by showing the step response to a prior $\eta = 3$ rad/s for both wheels:

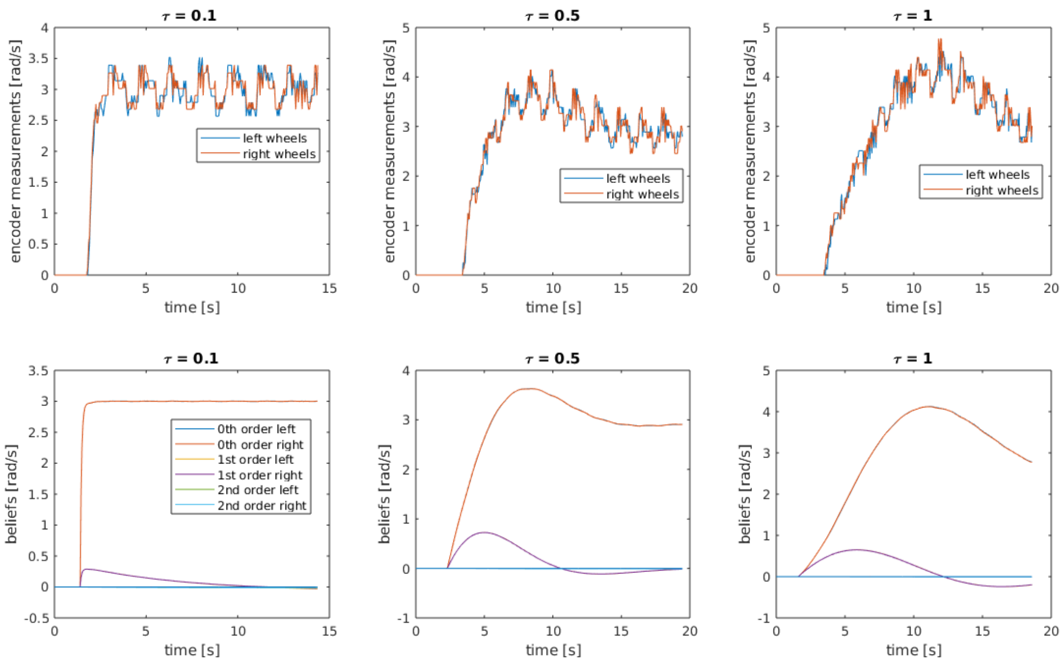


Figure 6.1: This graph shows the influence of changing the temporal parameter τ on the convergence of the beliefs or hidden states. Decreasing τ shows a faster convergence, but results in a high bias towards the prior seen in the way that the beliefs do not respond to the fluctuations in the encoder measurements

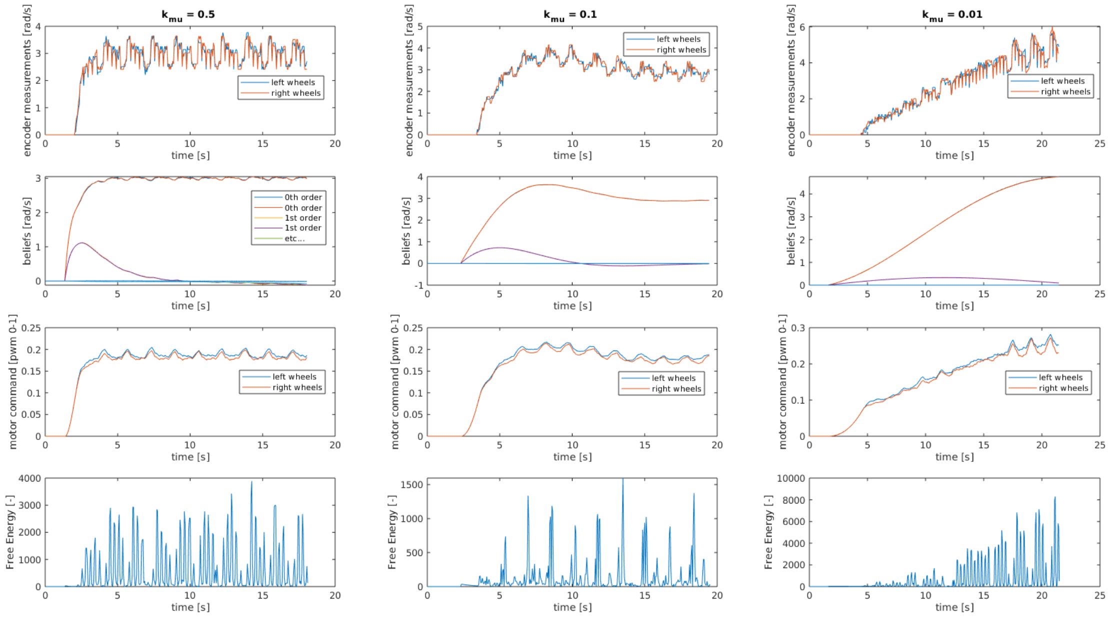


Figure 6.2: This graph shows the influence of κ_x on the convergence of the beliefs or hidden state. Increasing the learning rate κ_x will increase the convergence, but will also increase the influence of the observations on the beliefs. This is showing in the left-most graphs, where the fluctuations of the observations gets translated to the beliefs, but in the right graphs those fluctuations of the belief is not present.

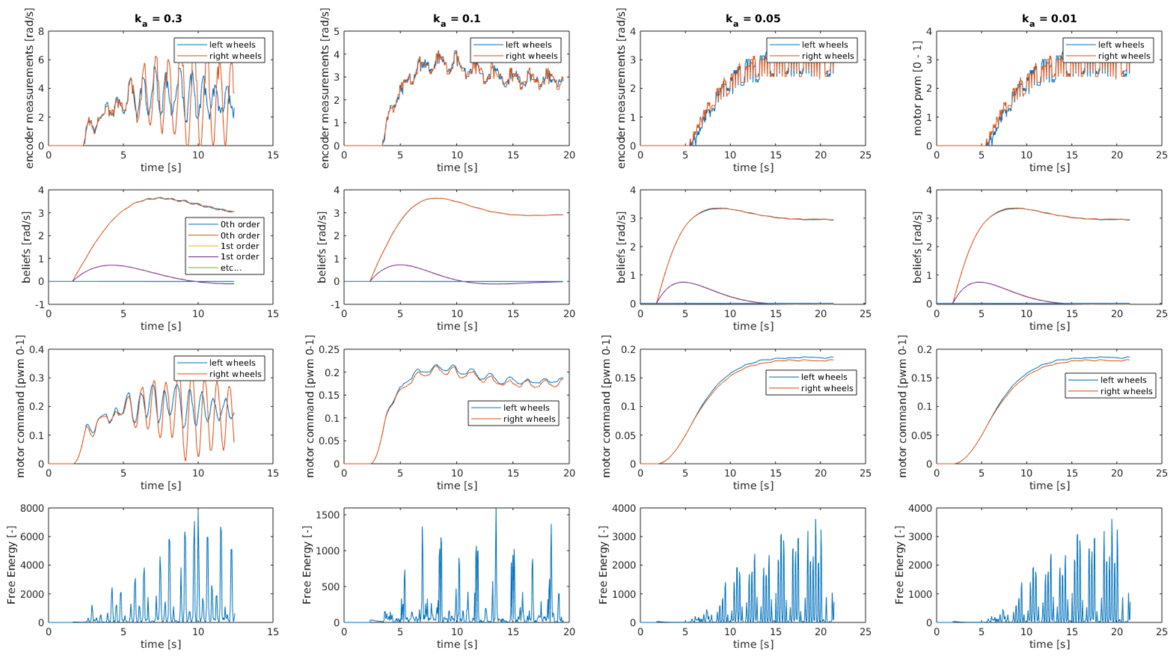


Figure 6.3: This graph shows the influence of κ_a on the convergence of the actions towards the beliefs. A higher learning parameter κ_a results in a faster convergence, when comparing the encoder measurements to the beliefs. From these graphs can be concluded that the learning parameter for the action is the one that is the most prone to instabilities. A learning rate κ_a = 0.1 seems to give satisfactory results (encoder measurements following the beliefs well)

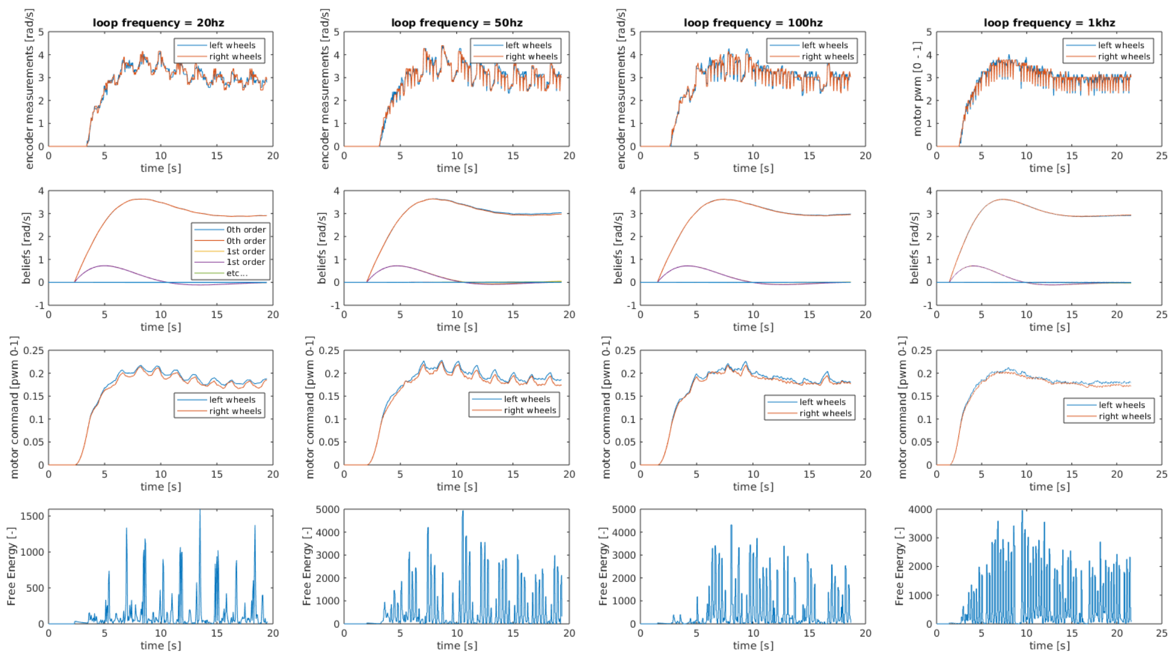


Figure 6.4: This graph shows the influence of the loop frequency of the active inference layer on the behaviour of the system. Starting from equal to measurement frequency (20Hz) the loop frequency is increased. The graphs show that the system can handle this no problem, and that it actually creates smoother graphs for the motor commands and the encoder measurements

A few things can be concluded for this specific tuning process:

- The learning rate k_x should be set more towards 0.5.
- The learning rate k_a around 0.1 should be fine
- The temporal parameter τ should be set more towards 0.1
- A loop frequency of 500Hz can be used, striking an arbitrary balance between the observed benefits of a higher loop frequency and the data processing done in matlab, as higher frequency means more data as it records a lot of states and their higher orders.
- Adjust the variances of the precision matrices upwards to get cleaner graphs for the free energy.

This cycle of logging the results of the independent changes of tuning parameters, whereafter the graphs are analyzed and the new parameters are estimated, can be repeated until satisfied with the result. The resulting graph for the independent tuning of the bottom layer is shown below:

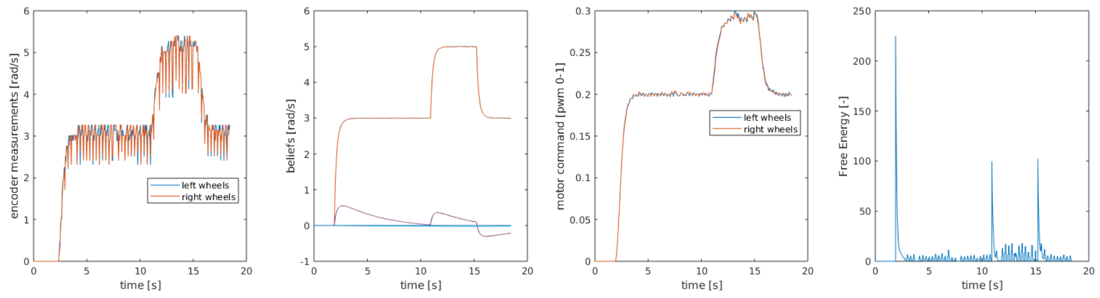


Figure 6.5: This graph shows the step response of the newly tuned bottom layer reacting to a prior of $\eta = 3$, after which the prior is changed to 5, and then back to 3. Additionally, the variances in the precision matrices are adjusted upwards, such that we can now clearly see the change in prior in the free energy graph.

6.2.2. Hierarchical tuning problem

The newly tuned bottom layer, as shown in Fig. 6.5 can be connected back to the top layer, after which the tuning of this layer can commence. It is already known that the bottom layer will converge to a prior, so the only things left to do is to get the hidden states μ_x of the top layer to converge faster towards the prior given by the user, and then get the causal states μ_v to converge faster towards the expected value based on those hidden states μ_x. The first part is not a problem, and can easily be done by adjusting κ_x and τ, a problem however arises when trying to converge the causal states μ_v. These causal states do not seem to easily want to converge towards the values expected from μ_x. An example is shown below in three subsequent figures where the learning rate κ_v is adjusted upwards to try to get the causal states to converge. The first figure has a $\kappa_v = 1.0$, the second figure a $\kappa_v = 3.0$ and the third has a $\kappa_v = 4.0$. The priors given to the system is a linear x velocity of 0.2 m/s, and an angular velocity of 0.4 rad/s. The resulting causal states μ_v are expected to be around 1 rad/s for the left wheel and 3 rad/s for the right wheel. The figures show a slow convergence for $\kappa_v = 1.0$, and a tripling of the learning rate to 3.0 having almost no effect on the convergence, where with other learning rates a doubling or tripling gives observable changes. At $\kappa_v = 4.0$ the whole systems even gets unstable.

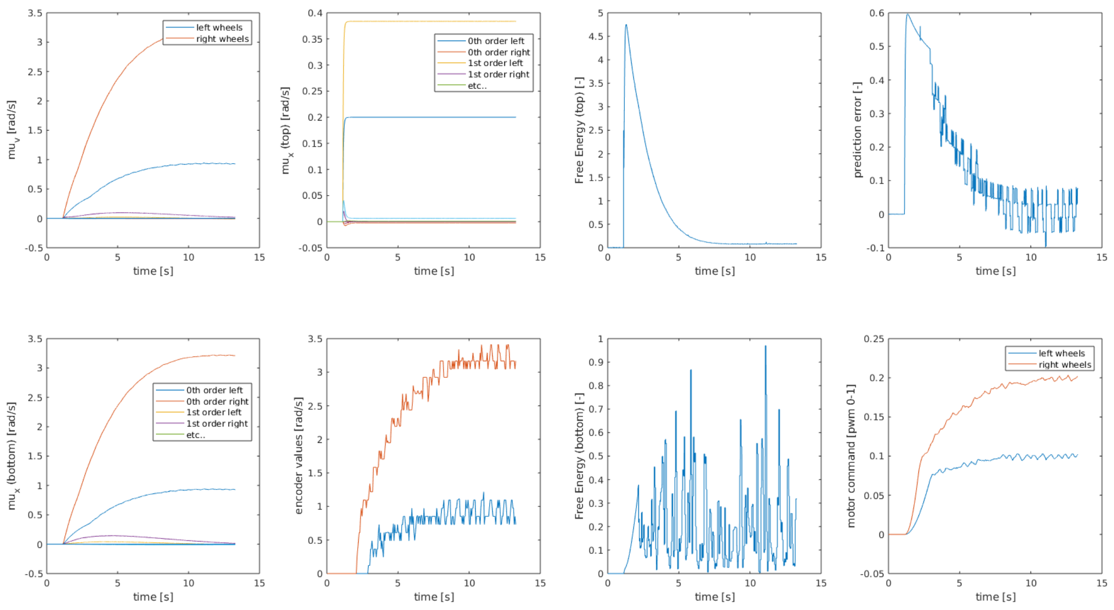


Figure 6.6: This graph shows the step response of a hierarchical system with a prior on linear velocity of 0.2 and prior on angular velocity of 0.4. This systems shows that in every aspect the system behaves fine, except for the convergence of the causal state μ_v. The convergence of the bottom layer towards this slow converged causal state is fast enough, as well as the convergence of the top layer hidden states μ_x towards the given priors

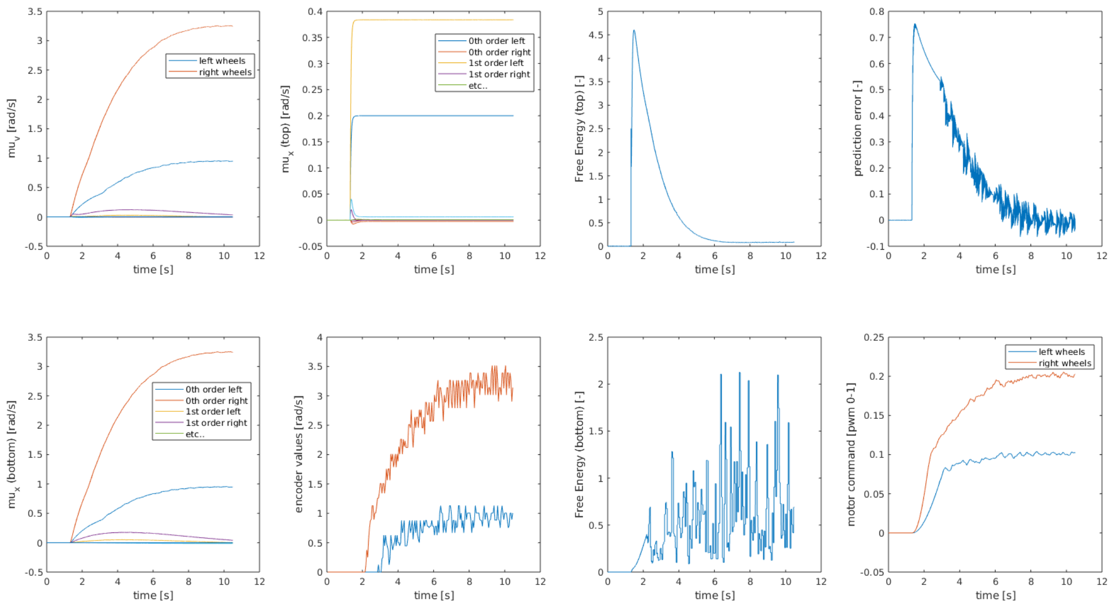


Figure 6.7: This graph has a κ_v of 3.0 as opposed to 1.0 of the previous graph. This indicates that a tripling of the learning rate does not give an observable difference in the convergence

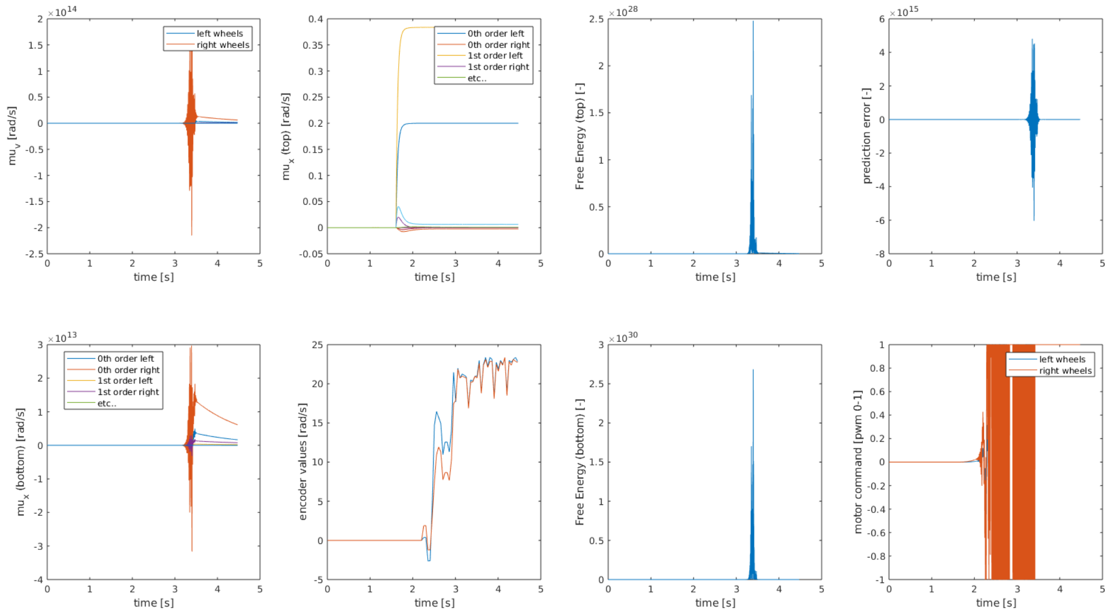


Figure 6.8: This graph has a κ_v of 4.0, and is completely not working. Other values higher than 4.0 also show this behaviour, meaning the learning rate can not be further increased than this.

The potential explanation of this problem could be found in the update rule for the causal state $\dot{\mu_v}$, as given in eq. 5.4. The update rule for the action takes into account the prediction error from the layer below. This prediction error has a negative impact on convergence, because when a change in causal state gets passed to the layer below as a prior it creates an error with the hidden state of that layer. This error gets passed back up again to the update rule of the causal state with instructions on how to change that error (second term between brackets in eq. 5.4). This part of the update rule thus works against the system and will try too keep the causal states equal to the hidden states of the bottom layer, rejecting any change. Increasing the learning rate κ_v will also increase the negative effect of the prediction error on the convergence, because this effect get also multiplied by that learning rate. In addition to this, if a slightly faster convergence is accomplished, this also means that the initial prediction error between the causal state and the hidden state of the bottom layer will be bigger.

A potential solution to this problem can be to add a coefficient (<1.0) to the prediction error term in order to decrease its negative influence on the convergence. This coefficient c is placed in the eq. 5.4 as follows:

$$\dot{\mu_v} = \mathcal{D}\mu_v - \kappa_v\left(\frac{\delta \mathcal{F}}{\delta \mu_v} - c\frac{\delta \varepsilon_x^{h-1}}{\delta \mu_v}\Pi_x \varepsilon_x^{h-1}\right) \tag{6.1}$$

Choosing this coefficient to be c = 0.1 for the initial tests gave the following graph:

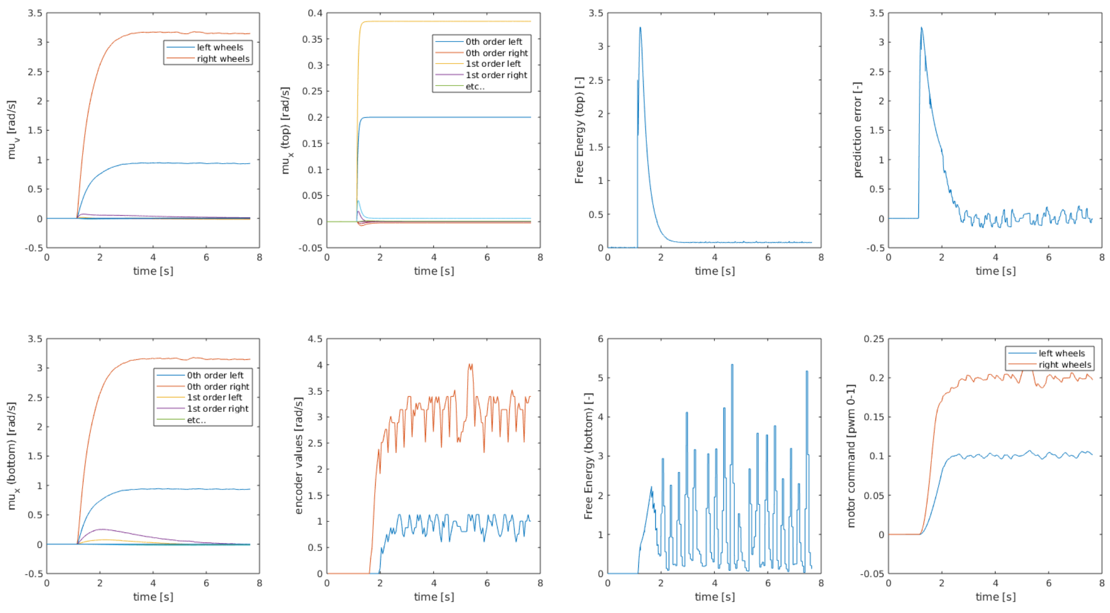


Figure 6.9: This graph has a κ_v of 4.0, but has an extra coefficient $c = 0.1$ in front of the prediction error term in the update rule for the causal state. Immediately it can be seen that the causal state μ_v converges much better, especially as the system was previously unstable on this learning rate

The addition of the extra parameter seems to be a success. Interesting to note is that with that parameter of $c = 0.1$ the same unstable behaviour returns at $\kappa_v = 40.0$ instead of the 4.0 without the added coefficient. Eventually, for the system, as compared in the next chapter, a coefficient of $c = 0.05$ is chosen.

6.2.3. Tuning lessons

The concluding lessons to be taken away from the tuning experiments are:

- The active inference loop can run without problems at a much higher frequency than the incoming observations.
- Tuning order from low level to high level seems to be desirable, only connecting the level above when the bottom is fully tuned.
- The causal state should be tuned last, after the layer below is fully tuned and the hidden states converge. This is because the causal state seems to give the most problems.
- The causal state will not converge without an extra parameter to help decrease the negative effect of the prediction error on the convergence.

7 Results

The purpose of this chapter is to show that the hierarchical implementation works in practice, and show how it stacks up in terms of performance against the non-hierarchical controller as presented in Chapter 4

7.1. Setup

As discussed in section 3.4, four experiments are carried out per controller, shown below in table 7.1, resulting in a total of 8 experiments. Pivoting consists of giving a prior for the angular velocity of $\dot{\theta} = 0.4$ and $\dot{x} = 0.0$, resulting in a rotation in place. Circle driving means giving a prior for the angular velocity of $\dot{\theta} = 0.4$ and linear velocity of $\dot{x} = 0.2$. On box means with the wheels suspended, resulting in a stationary robot. On ground means the robot is placed on the ground and is free to move.

	on box	on ground
pivoting	exp. 1	exp. 3
circle driving	exp. 2	exp. 4

Table 7.1: the different experiments done for both controllers

7.2. Non-hierarchical results

The results of the experiments for the non-hierarchical controller are shown in the graphs below:

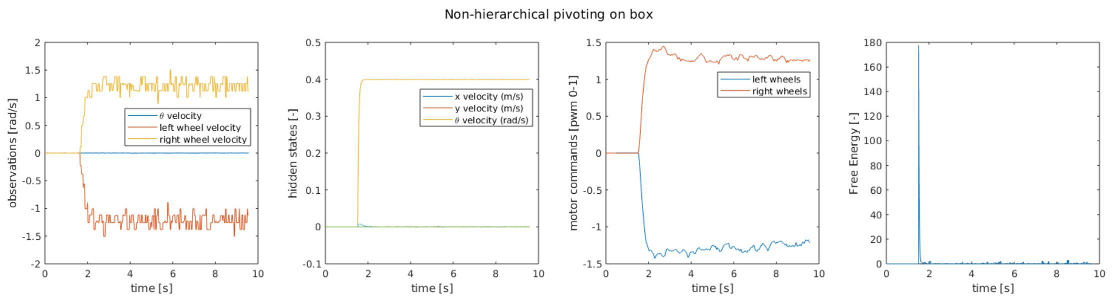


Figure 7.1: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.0$ and $\dot{\theta} = 0.4$, resembling pivoting in place. This is experiment is done with the wheels suspended, and therefore a stationary robot, consequently not measuring changes in the angular velocity in the first graph. The second graph shows a fast convergence of the hidden states towards the prior. The free energy, shown in the fourth graph, is seen to have a spike at the initial command, whereafter the free energy is quickly brought back to (and kept at) zero.

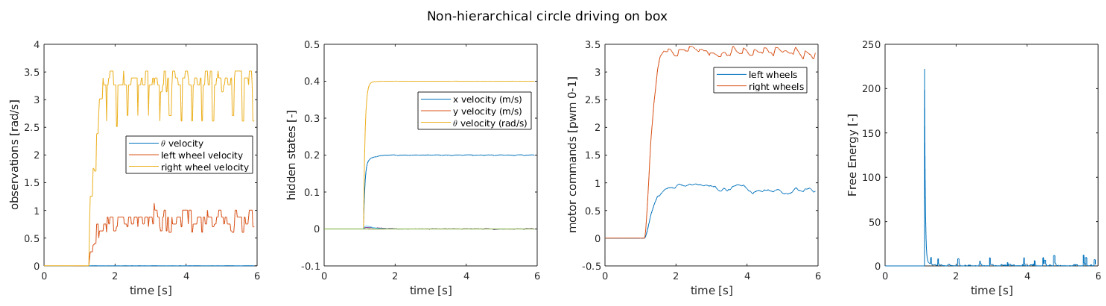


Figure 7.2: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.2$ and $\dot{\theta} = 0.4$, resembling circle driving. This is experiment is done with the wheels suspended, and therefore a stationary robot, consequently not measuring changes in the angular velocity in the first graph. The second graph shows a fast convergence of the hidden states towards the prior. The third graph shows the difference in motor commands as the robot is cornering instead of pivoting. The fourth graph shows the free energy spike at the initial command, whereafter it is quickly brought back to (and kept at) zero.

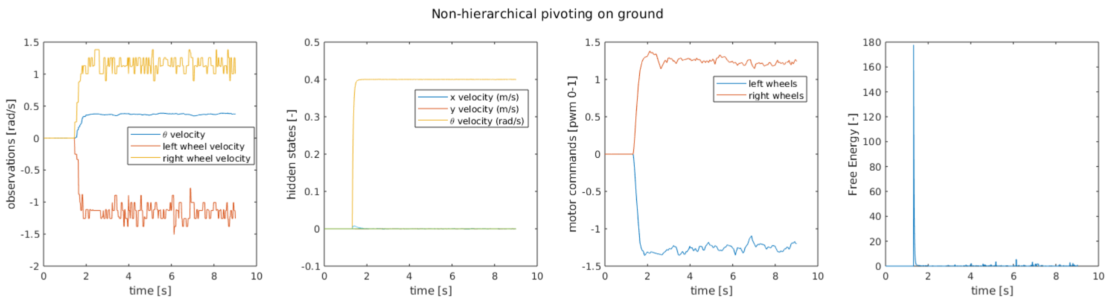


Figure 7.3: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.0$ and $\dot{\theta} = 0.4$, resembling pivoting in place. This is experiment is done on the ground, with the robot moving. As can be seen in the first graph, the robot is actually rotating, indicated by the blue line. The second, third and fourth graph are very similar to the on box experiment indicating no influence on performance.

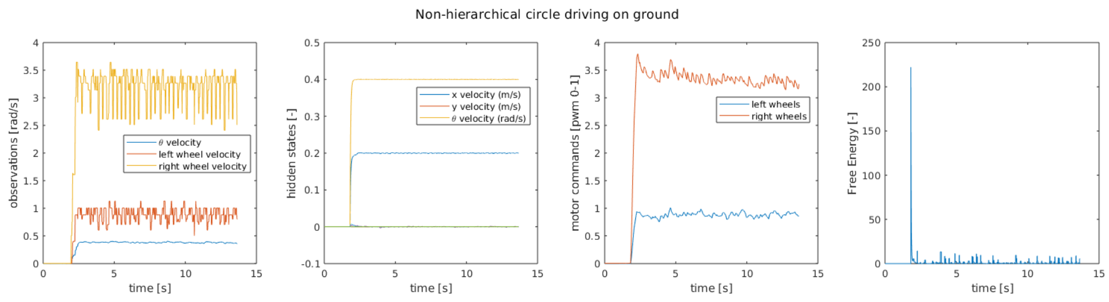


Figure 7.4: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.2$ and $\dot{\theta} = 0.4$, resembling circle driving. This is experiment is done on the ground, with the robot moving. The noise in the encoder signal of the outside wheel, as indicated in yellow in the first graph, seems to intensify. These oscillations however are not noticeable by eye. The performance is on par with the on box experiment.

From the graphs above it can be concluded that the non-hierarchical implementation was successful, and looking at the smoothness of the IMU signal, makes smooth turns as well. There is also no difference in performance between driving with the wheels suspended or on the ground.

7.3. Hierarchical results

The results of the experiments for the hierarchical controller are shown in the graphs below:

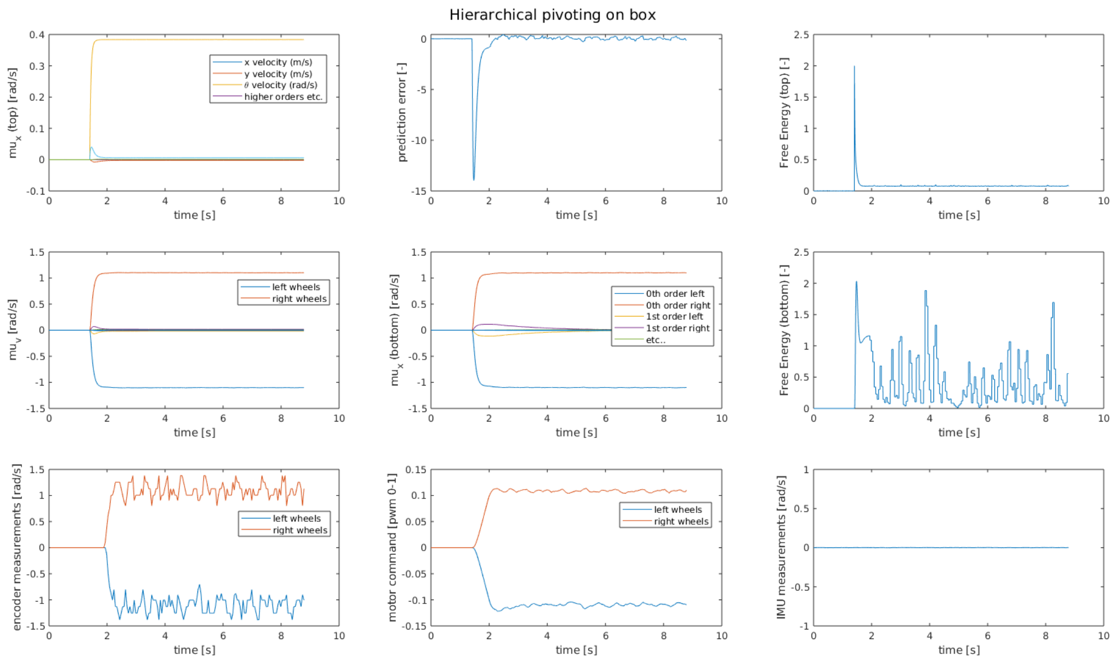


Figure 7.5: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.0$ and $\dot{\theta} = 0.4$, resembling pivoting in place. This is experiment is done with the wheels suspended, and therefore a stationary robot, consequently not measuring a change in angular velocity. As can be seen in the first graph, the hidden states of the top layer converge quickly but not to exactly the right value. This seems to be attributed to the robot being on a box, measuring zero for angular velocity, as this error is solved when the robot is put on the ground. There is however no explanation as to why this is not the case for the non-hierarchical controller. Another thing to notice is the free energy for the high level having a distinguishable single peak for the initial command, but the low level having multiple peaks of roughly the same height as the initial one, possibly indicating a more gradual change in prior.

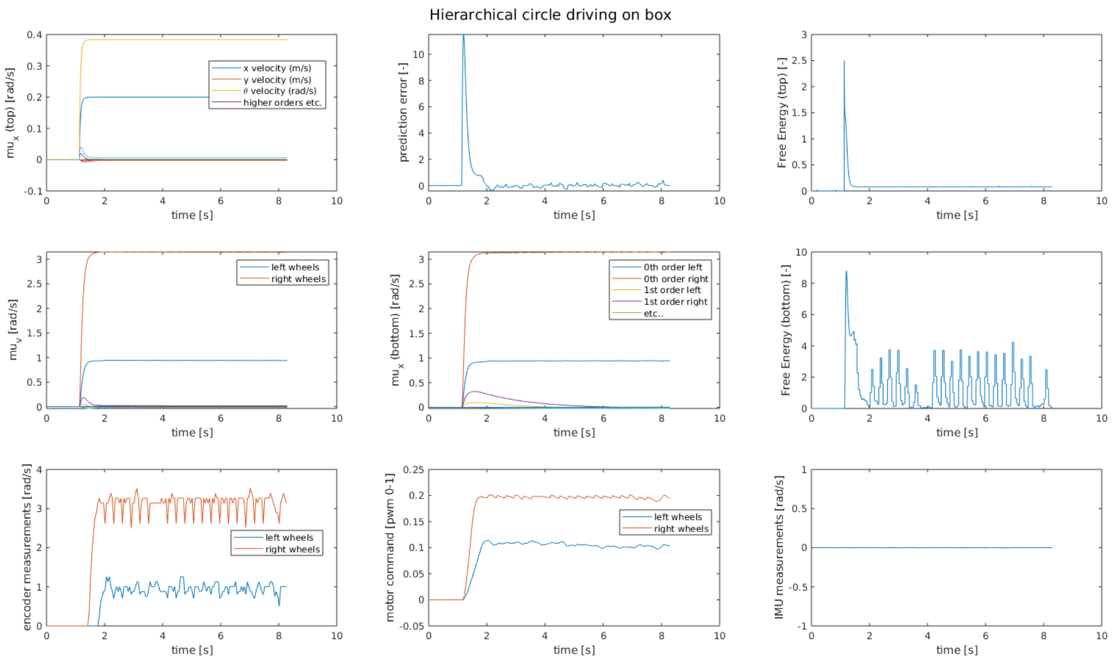


Figure 7.6: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.2$ and $\dot{\theta} = 0.4$, resembling circle driving. This is experiment is done with the wheels suspended, and therefore a stationary robot, consequently not measuring a change in angular velocity. The circle driving experiment does not differ much from the pivoting experiment for the on-box situation, still showing a slight error in angular velocity for the hidden states of the top level.

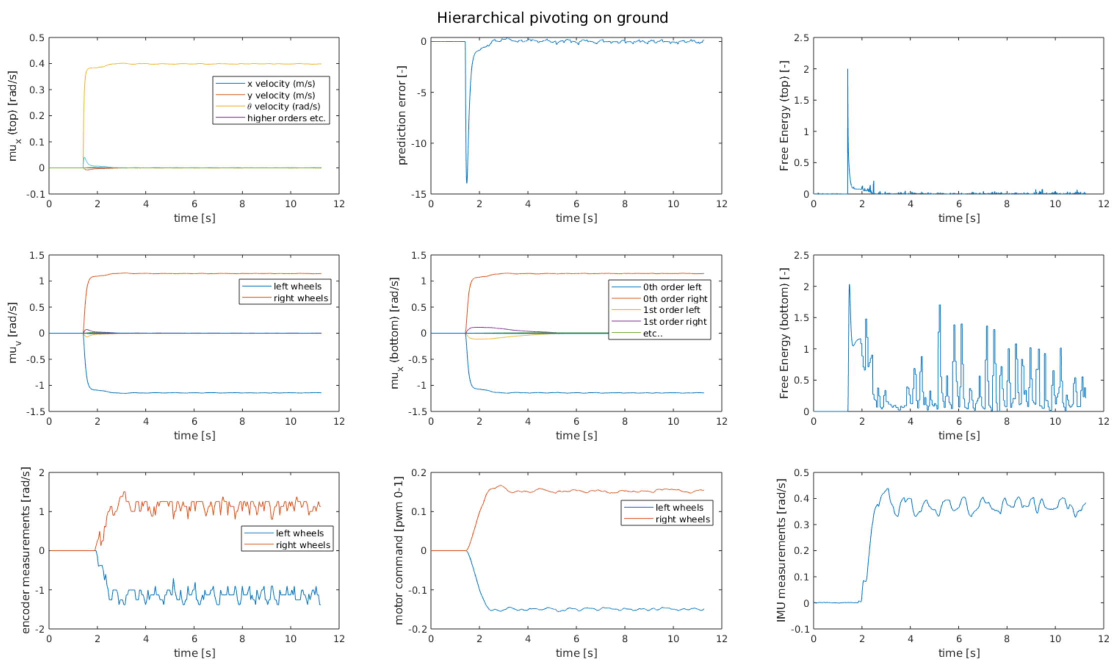


Figure 7.7: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.0$ and $\dot{\theta} = 0.4$, resembling pivoting in place. This is experiment is done on the ground, with the robot moving. The slight error in the hidden state for the angular velocity is fixed in the ground experiment, indicating the problem was probably due to the robot not being able to move. The step response on the encoder measurements, as seen in the bottom-left graph, is slightly slower than the non-hierarchical version. Nonetheless it converges to a quite stable rotation, with only slight oscillations, as indicated by the bottom-right graph.

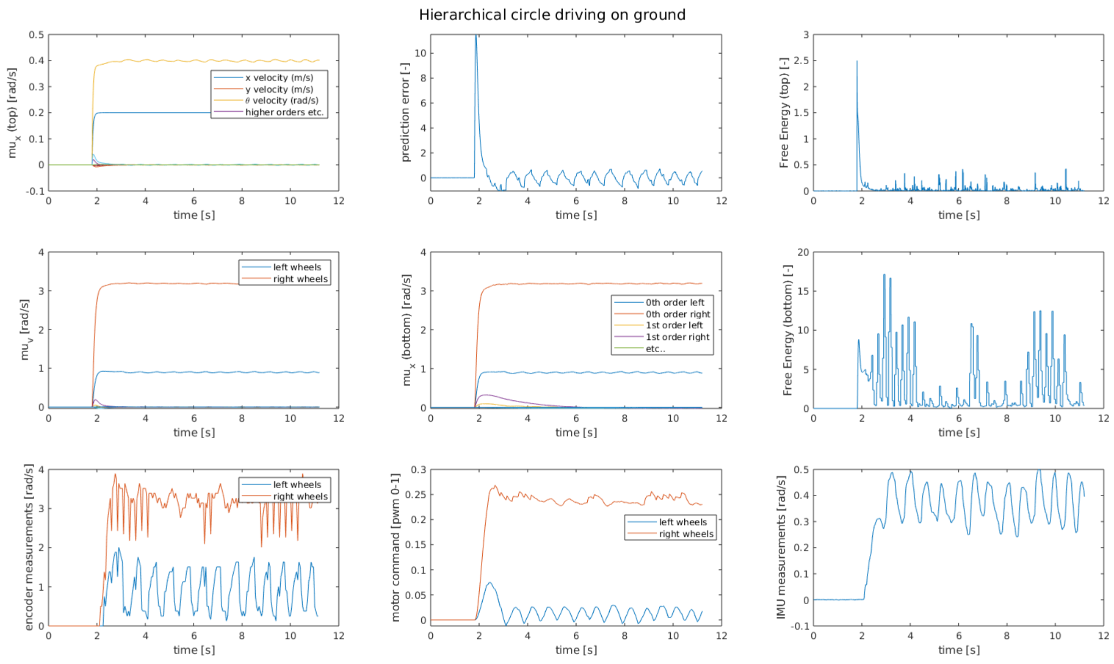


Figure 7.8: These graphs show the states of a non-hierarchical active inference controller during a step response towards a prior on $\dot{x} = 0.2$ and $\dot{\theta} = 0.4$, resembling circle driving. This is experiment is done on the ground, with the robot moving. This is where the lack of performance is the most clear. The inner wheel oscillates greatly in speed, as can be seen in the bottom-left graph. This makes it such that the turning is not observed to be smooth, as indicated by the bigger oscillations in the bottom-right graph.

From the graphs above it can be concluded that, when suspended on a box, the hierarchical controller is very similar to the non-hierarchical controller in performance. However, when putting the robot on the ground, and especially during the circle driving experiment, the hierarchical active inference controller struggles a lot to turn smoothly. The angular velocity oscillates around the desired value, and this behaviour is highlighted more when driving in a circle as compared to pivoting in place.

This behaviour can probably be attributed to the stick-slip behaviour of the wheel-ground interaction. The stick-slip can be better handled by the non-hierarchical version as this benefits from the PID wheel speed control having access to high frequency encoder and/or motor measurements. The hierarchical version only has access to the 20Hz feedback that is supplied within ROS. Values of the learning rate κ_a has also been played with to see if this oscillating behaviour can be dampened, but to no avail. The difference in encoder feedback makes it such that it is too premature to conclude that active inference is less suited to handle low level control, although it does seem that way from the results. Another interesting observation is the slight error in the settling value for the hidden state of the angular velocity, when the robot is not able to move. This error probably will increase when the observation of the angular velocity is tuned to have a bigger impact. This might be the reason why this error was only noticed in the hierarchical version, as the observation of the angular velocity was probably tuned to have more influence on the update rule of the hidden state than it did for the non-hierarchical version. Lastly, the free energy spikes that can be clearly identified in the non-hierarchical version, and the highest layer of the hierarchical version, seem to be less prominent in the lower level, which would make sense, as the top level command has a more indirect influence on the free energy of the lower level.

8 Conclusion

This first goal of this thesis was to show how the structure of an hierarchical active inference controller expands from its non-hierarchical version. This is done in a ’building block’ way, showing how multiple layers of active inference can be stacked. A novel mapping was added from the hidden states towards the causal states, serving as the prior for the causal states. In this work, the causal state update loop is joined together with an active inference layer, making hierarchy only possible vertically, meaning only one instance of active inference can be stacked on top of another. In order to make the structure suitable to handle different branches in hierarchy (horizontal expansion), the causal state has to be taken out of the active inference layer and its update loop, and create its own layer between the different layers of active inference, independently updating its states.

The second, and arguably most important goal, was to get a ’working’ solution of both controllers. This was achieved, proving the implementation structure, as described in this work, was successful and because of its modular nature could also be successful for other applications.

A big part of getting the controller to work was managed by the third goal, the tuning of both controllers. During tuning it was discovered that the hierarchical implementation has trouble with the convergence on the causal state, mainly due to the identified negative influence of the prediction error passed from below to the update rule of this causal state. A potential solution in the form of a coefficient in front of the prediction error part in the update rule equation was introduced. This work does not claim that this is the best solution to that problem, but it presents a potential fix. The best way to tune a hierarchical system seems to be from bottom to top, independently tuning the layer(s) below and only connecting the layer(s) above when the layer below is fully tuned. Because during tuning the main difficulties arise with getting the causal states to converge, this should be tuned only after to layer above and below are fully tuned. Tuning experiments further showed that the active inference layers have no problem running at much higher frequencies than the measuring frequency. Choosing and tuning precision matrices are done in a such a way the free energy graphs look clean, resulting in a situation where for convenience the variances are chosen so high that the higher orders almost have no influence anymore on the system. Then later on if the system works with the lower orders, the variances can be adjusted downwards to bring in more influence of the higher orders in the update of states. One reason behind this is the fact that the generalized measurements get really noisy for the higher orders, and you just be better off not using them. These higher orders blowing up is due to the generalization technique (backward differentiation) that is being used. Better generalization techniques might suffer less from this issue.

The performance of the hierarchical active inference controller is compared to the non-hierarchical version, being the fourth and final sub-goal. In the scenario with the wheels suspended, both controllers are similar in performance, but when the wheels touch the ground, the hierarchical controller suffers from stick-slip issues. This can probably be attributed to the very low feedback frequency on the wheel speeds that are available to the hierarchical system, as opposed to the MCU-level PID control on which the non-hierarchical system relies for its wheel speed control. This makes it not really a fair comparison, but it does show that both controllers do work fine and that active inference, although it still needs further investigation with higher feedback frequencies, probably is less suited for the very low-level control. In the on-box experiments for the hierarchical controller, it was also noticed that the hidden states of the angular velocity does not fully converge to the prior, probably attributed to the inability of the robot to move. However, this did not occur in the non-hierarchical version, which might be explained by the possibility that observations are tuned to have less influence in the update rule of the hidden states. Lastly, the top-level commands can be distinguished very well as peaks in the free energy graphs of the non-hierarchical controller and the top level of the hierarchical version, but these peaks showed to be less identifiable in the bottom active inference layer, as might be expected.

This work has delivered a proven codebase for a flexible and modular implementation of hierarchical active inference in the ROS framework. Hopefully this can be used and improved upon to help further the research on hierarchical active inference for robotics.

9 Discussion

9.1. Simplifications and assumptions

The first simplification is made with choosing the generative function $f(\mu_x)$ this function contains the dynamics of the system, in this work it is chosen to not implement a dynamic model, but instead only use attractor dynamics that encodes the preference to move towards the prior.

The noise on the system is assumed to be independent, and therefore only the diagonals of the precision matrices Π are filled.

For practicality, these precision matrices Π are tuned in such a way that higher orders of the generalized beliefs do not have much influence anymore on the system.

The system only keeps track of 3 higher orders of the states, meaning an embedding order of 3. Although not really a big simplification, as these higher orders are probably tuned away as well, but keeping higer orders means using more information of the system. Friston [5] mentioned that we should probably strive to an embedding order of 6.

The forward model $\frac{\delta y}{\delta a}$ is only used for the first order. For higher orders this is assumed zero. The reason for this is that you cannot accurately tell what the higher orders on the wheels speeds, like acceleration and jerk will do with only a change in voltage.

The generalization technique used, backwards differentiation, is a very simple way of calculating the higher order derivatives of a system based on past measurements. The problem with this technique is that, besides not giving the best representation of current higher orders because it says something about past higher orders, the higher orders also diverge very quickly and get blown up to a point where variances of over the millions needs to be used to not let it affect the system.

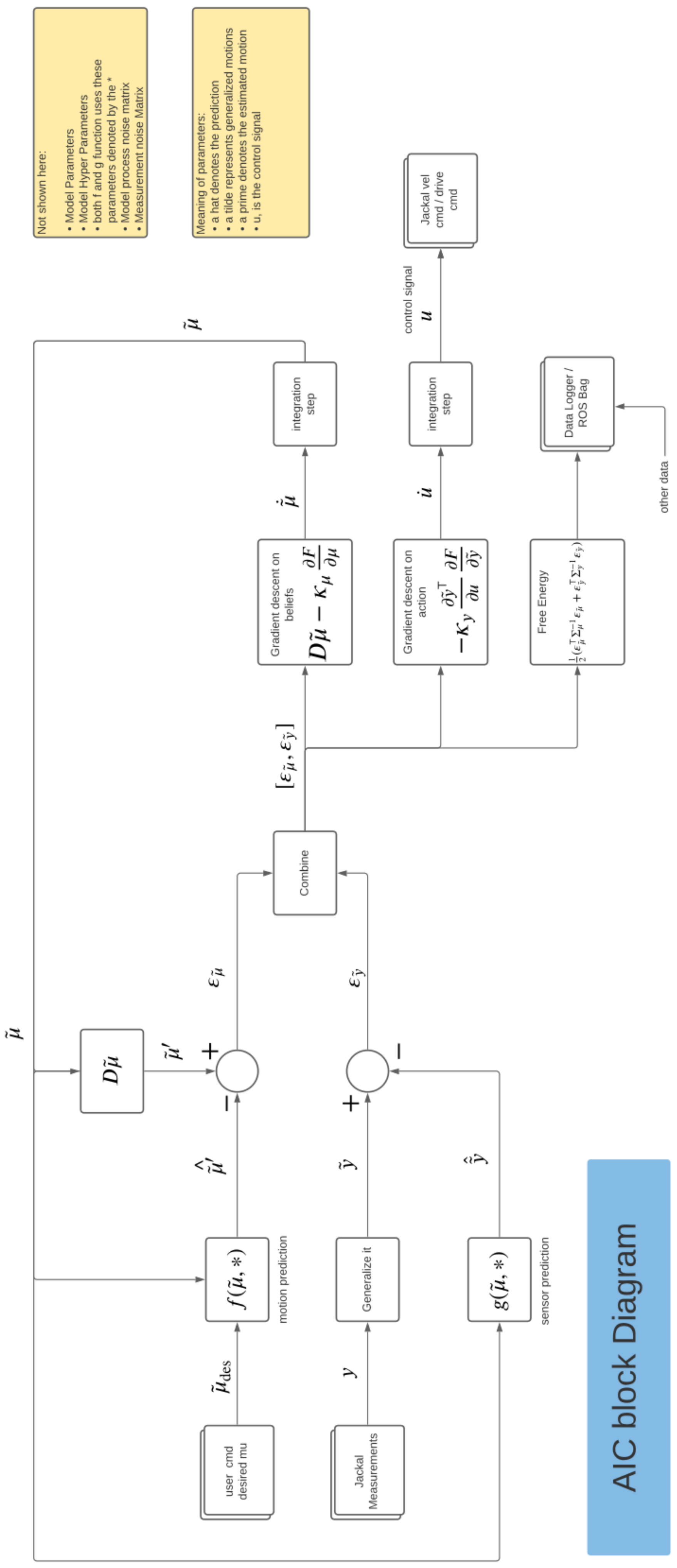
The hierarchical system is structured in such a way that it can only handle 1-to-1 vertical expansion. This is done mostly because of when the causal state can get incorporated in an active inference layer, the system needs less separate loops to run.

9.2. Further research recommendations

Based on the simplifications, it would be interesting to see how an active inference controller behaves with a complex dynamic model, instead of only attractor dynamics. The noise could also be further investigated, together with a strategy for choosing, calculating or tuning the precision matrices. It would also be interesting to see how this hierarchical system performs with a better generalization technique, based on a model, instead of only past values. If the generalization is better, then maybe the precision matrices will not need to have very high variances to 'tune away' the data. It will then also be interesting to see how these higher orders would get tuned back into the system and on a system or experiment where the benefits of keeping track of those higher orders can be clearly pointed out. The implemented hierarchy structure can also be expanded to more levels hierarchy, for which a suitable application needs to be found, for example on top of this controller, an inference layer for the position of the robot in the global frame, and another inference layer on top of that tracking objects in the environment, with another layer on top taking decisions on where to go next. This implementation can probably be done with the current structure, but the structure could also be adapted to handle horizontal expansion as well, creating a web of active inference layers like neurons in a brain, instead of just a stack. A suitable practical implementation for this still needs to be found.

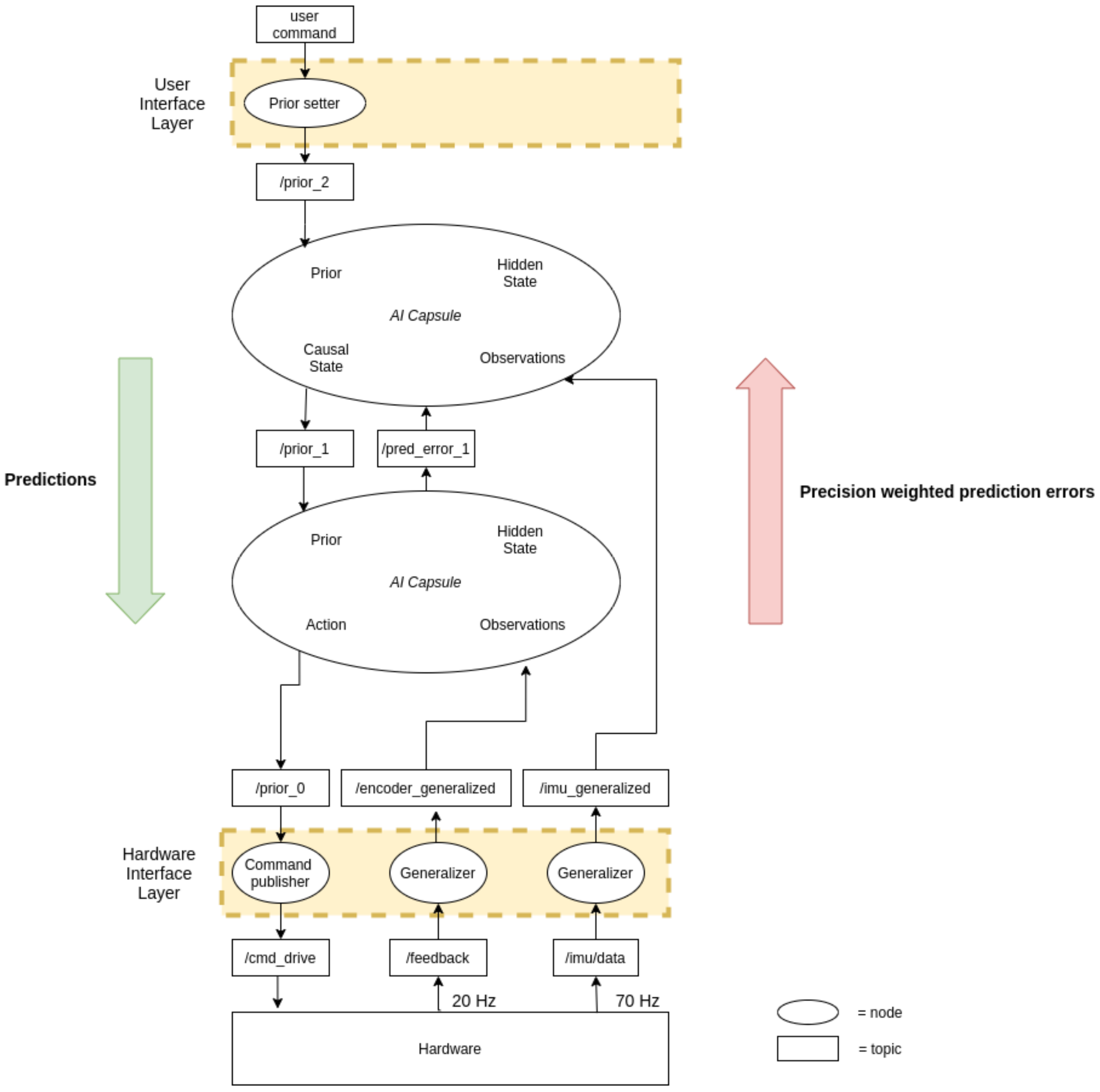
A
Appendix A

The next page includes a full-page view of the controller structure of the non-hierarchical active inference controller that has been implemented as a collective effort between 4 masters students.


AIC block Diagram
Not shown here:
• Model Parameters
• Model Hyper Parameters
• both f and g function uses these parameters denoted by the *
• Model process noise matrix
• Measurement noise Matrix
Meaning of parameters:
• a hat denotes the prediction
• a tilde represents generalized motions
• a prime denotes the estimated motion
• u, is the control signal
user cmd desired mu
$\tilde{\mu}_{\text{des}}$
$f(\tilde{\mu}, *)$
motion prediction
$\hat{\tilde{\mu}}'$
Jackal Measurements
y
Generalize it
$\tilde{y}$
$g(\tilde{\mu}, *)$
sensor prediction
$\hat{\tilde{y}}$
$\tilde{\mu}$
$D\tilde{\mu}$
$\tilde{\mu}'$
$+$
$-$
$+$
$-$
$\varepsilon_{\tilde{\mu}}$
$\varepsilon_{\tilde{y}}$
Combine
$[\varepsilon_{\tilde{\mu}}, \varepsilon_{\tilde{y}}]$
Gradient descent on beliefs
$D\tilde{\mu} - \kappa_{\mu} \frac{\partial F}{\partial \mu}$
$\dot{\tilde{\mu}}$
integration step
$\tilde{\mu}$
Gradient descent on action
$-\kappa_{y} \frac{\partial \tilde{y}}{\partial u}^{\top} \frac{\partial F}{\partial \tilde{y}}$
$\dot{u}$
integration step
control signal
u
Jackal vel cmd / drive cmd
Free Energy
$\frac{1}{2}(\varepsilon_{\tilde{\mu}}^{\top} \Sigma_{\tilde{\mu}}^{-1} \varepsilon_{\tilde{\mu}} + \varepsilon_{\tilde{y}}^{\top} \Sigma_{\tilde{y}}^{-1} \varepsilon_{\tilde{y}})$
Data Logger / ROS Bag
other data

B

Appendix B


user command
User Interface Layer
Prior setter
/prior_2
Prior
Hidden State
AI Capsule
Causal State
Observations
Predictions
Precision weighted prediction errors
/prior_1
/pred_error_1
Prior
Hidden State
AI Capsule
Action
Observations
/prior_0
/encoder_generalized
/imu_generalized
Hardware Interface Layer
Command publisher
Generalizer
Generalizer
/cmd_drive
/feedback
/imu/data
20 Hz
70 Hz
Hardware
= node
= topic

Bibliography

[1] Georgia Anousaki and Kostas Kyriakopoulos. “A Dead-reckoning Scheme for Skid-steered Vehicles in Outdoor Environments.” In: *Proceedings - IEEE International Conference on Robotics and Automation* (Jan. 2004), pp. 580–585. DOI: `10.1109/ROBOT.2004.1307211`.

[2] Mohamed Baioumy et al. “Active Inference for Integrated State-Estimation, Control, and Learning”. In: *arXiv:2005.05894* (May 2020).

[3] Manuel Baltieri and Christopher Buckley. *An active inference implementation of phototaxis*. Sept. 2017. DOI: `10.7551/ecal\_a\_011`. URL: `http://sro.sussex.ac.uk/id/eprint/71872/`.

[4] Daniel Felleman and David Essen. “Distributed Hierarchical Processing in the Primate Cerebral Cortex”. In: *Cerebral Cortex* 1 (Jan. 1991).

[5] Karl Friston. “Friston, K.J.: The free-energy principle: a unified brain theory? Nat. Rev. Neurosci. 11, 127-138”. In: *Nature reviews. Neuroscience* 11 (Feb. 2010), pp. 127–38. DOI: `10.1038/nrn2787`.

[6] Karl Friston. “Hierarchical Models in the Brain”. In: *PLoS computational biology* 4 (Dec. 2008), e1000211. DOI: `10.1371/journal.pcbi.1000211`.

[7] Sherin Grimbergen. “Active Inference for State Space Models: A Tutorial”. In: *TU Delft repository* (May 2019).

[8] I Hijne. “Generalised Motions in Active Inference by Finite Differences”. In: *TU Delft repository* (Aug. 2020).

[9] Stefan Kiebel, Jean Daunizeau, and Karl Friston. “A Hierarchy of Time-Scales and the Brain”. In: *PLoS computational biology* 4 (Dec. 2008), e1000209. DOI: `10.1371/journal.pcbi.1000209`.

[10] Pablo Lanillos and G. Cheng. “Active inference with function learning for robot body perception”. In: *International Workshop on Continual Unsupervised Sensorimotor Learning (ICDL-Epirob)* (2018).

[11] Anthony Mandow et al. “Experimental kinematics for wheeled skid-steer mobile robots”. In: *Conference: Intelligent Robots and Systems* (Dec. 2007), pp. 1222–1227. DOI: `10.1109/IROS.2007.4399139`.

[12] Beren Millidge. *Combining Active Inference and Hierarchical Predictive Coding: A Tutorial Introduction and Case Study*. Mar. 2019. DOI: `10.31234/osf.io/kf6wc`.

[13] S.A.A. Moosavian and Arash Kalantari. “Experimental Slip Estimation for Exact Kinematics Modeling and Control of a Tracked Mobile Robot”. In: *Conference: Intelligent Robots and Systems* (Oct. 2008), pp. 95–100. DOI: `10.1109/IROS.2008.4650798`.

[14] G. Oliver, Pablo Lanillos, and G. Cheng. “Active inference body perception and action for humanoid robots”. In: *ArXiv* abs/1906.03022 (2019).

[15] Thomas Parr and Karl Friston. “Active inference and the anatomy of oculomotion”. In: *Neuropsychologia* 111 (Mar. 2018), pp. 334–343. DOI: `10.1016/j.neuropsychologia.2018.01.041`.

[16] C. Pezzato, R. Ferrari, and C. H. Corbato. “A Novel Adaptive Controller for Robot Manipulators Based on Active Inference”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2973–2980. DOI: `10.1109/LRA.2020.2974451`.

[17] G. Pezzulo, F. Rigoli, and K. J. Friston. “Hierarchical Active Inference: A Theory of Motivated Control”. In: *Trends in cognitive sciences* 111 (Apr. 2018), pp. 294–306. DOI: `10.1016/j.tics.2018.01.009`.

[18] Léo Pio-Lopez et al. “Active inference and robot control: A case study”. In: *Journal of the Royal Society, Interface* 13 (Sept. 2016). DOI: 10.1098/rsif.2016.0616.

[19] Rajesh Rao and Dana Ballard. “Predictive Coding in the Visual Cortex: a Functional Interpretation of Some Extra-classical Receptive-field Effects”. In: *Nature neuroscience* 2 (Feb. 1999), pp. 79–87. DOI: 10.1038/4580.

[20] Neal Seegmiller. “Dynamic Model Formulation and Calibration for Wheeled Mobile Robots”. PhD thesis. Pittsburgh, PA: Carnegie Mellon University, Oct. 2014.

[21] V. van Vucht. “Hierarchical Active Inference for Continuous Robot Control: Goal-Directed Hierarchical Models for ActiveInference Agents”. In: *TU Delft repository* (Jan. 2020).

[22] Tianmiao Wang et al. “Analysis and Experimental Kinematics of a Skid-Steering Wheeled Robot Based on a Laser Scanner Sensor”. In: *Sensors* 15 (May 2015), pp. 9681–9702. DOI: 10.3390/s150509681.