Active Inference and Behavior Trees for Reactive Action Planning and Execution in Robotics

Pezzato, Corrado; Corbato, Carlos Hernandez; Bonhof, Stefan; Wisse, Martijn

DOI
10.1109/TRO.2022.3226144

Publication date
2023

Document Version
Final published version

Published in
IEEE Transactions on Robotics

Citation (APA)
Pezzato, C., Corbato, C. H., Bonhof, S., & Wisse, M. (2023). Active Inference and Behavior Trees for Reactive Action Planning and Execution in Robotics. *IEEE Transactions on Robotics*, *39*(2), 1050-1069. https://doi.org/10.1109/TRO.2022.3226144

Active Inference and Behavior Trees for Reactive Action Planning and Execution in Robotics

Corrado Pezzato, Carlos Hernández Corbato, Stefan Bonhof, and Martijn Wisse

***Abstract*—In this article, we propose a hybrid combination of active inference and behavior trees (BTs) for reactive action planning and execution in dynamic environments, showing how robotic tasks can be formulated as a free-energy minimization problem. The proposed approach allows handling partially observable initial states and improves the robustness of classical BTs against unexpected contingencies while at the same time reducing the number of nodes in a tree. In this work, we specify the nominal behavior offline, through BTs. However, in contrast to previous approaches, we introduce a new type of leaf node to specify the desired state to be achieved rather than an action to execute. The decision of which action to execute to reach the desired state is performed online through active inference. This results in continual online planning and hierarchical deliberation. By doing so, an agent can follow a predefined offline plan while still keeping the ability to locally adapt and take autonomous decisions at runtime, respecting safety constraints. We provide proof of convergence and robustness analysis, and we validate our method in two different mobile manipulators performing similar tasks, both in a simulated and real retail environment. The results showed improved runtime adaptability with a fraction of the hand-coded nodes compared to classical BTs.**

***Index Terms*—Active inference, behavior trees (BT), biologically inspired robots, free-energy principle, mobile manipulators, reactive action planning.**

I. Introduction

Deliberation and reasoning capabilities for acting are crucial parts of online robot control, especially when operating in dynamic environments to complete long-term tasks. Over the years, researchers developed many task planners with various degrees of optimality, but little attention has been paid to actors [1], [2], i.e., algorithms endowed with reasoning and deliberation tools during plan execution. The authors in [1] and [2] advocate a change in focus, explaining why this lack of actors could be one of the main causes of the limited spread of automated planning applications. Authors in [3], [4], [5] proposed the use of BTs as graphical models for more reactive task execution, showing promising results. Other authors also tried to answer the call for actors [6], [7], but there are still open challenges to be addressed. These challenges have been identified and highlighted by many researchers, and can be summarized as in [1] and [4] in two properties that an actor should possess.

1) *Hierarchical deliberation:* each action in a plan may be a task that an actor may need to further refine online.
2) *Continual online planning and reasoning:* an actor should monitor, refine, extend, update, change, and repair its plans throughout the acting process, generating activities dynamically at run time.

Actors should not be mere action executors then, but they should be capable of intelligently taking decisions. This is particularly useful for challenging problems, such as mobile manipulation in dynamic environments, where actions planned offline are prone to fail. In this article, we consider mobile manipulation tasks in a retail environment with a partially observable initial state. We propose an actor based on active inference. Such an actor is capable of following a task planned offline while still being able to take autonomous decisions at run-time to resolve unexpected situations.

Active inference is a neuroscientific theory that has recently shown its potential in control engineering and robotics [8], [9], [10], [11], particularly in real-world experiments for low-level adaptive control [12], [13]. Active inference describes a biologically plausible algorithm for perception, action, planning, and learning. This theory has been initially developed for continuous processes [14], [15], [16], where the main idea is that the brain's cognition and motor control functions could be described in terms of *free-energy minimization* [17]. In other words, we, as humans, take actions in order to fulfill prior expectations about a *desired prior* sensation [18]. Active inference has also been extended to Markov decision processes for discrete decision-making [19], recently gathering more and more interest [20], [21], [22], [23]. In this formulation, active inference is proposed as a unified framework to solve the exploitation–exploration dilemma by acting to minimize the free-energy. Agents can solve complicated problems once provided a *context sensitive prior* about preferences. Probabilistic beliefs about the state of the world are built through Bayesian inference, and a finite horizon plan is selected in order to maximize the evidence for a model that is biased toward the agent's preferences. At the time of writing, the use of discrete active inference for symbolic action planning is limited to low-dimensional and simplified

Manuscript received 13 June 2022; revised 19 October 2022; accepted 15 November 2022. Date of publication 2 January 2023; date of current version 5 April 2023. This work was supported by Ahold Delhaize. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors. This article was recommended for publication by Associate Editor J. Bohg and Editor W. Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Corrado Pezzato.)*

The authors are with the Department of Cognitive Robotics, TU Delft, 2628 CD Delft, The Netherlands (e-mail: c.pezzato@tudelft.nl; c.h.corbato@tudelft.nl; s.d.bonhof@gmail.com; m.wisse@tudelft.nl).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2022.3226144.

Digital Object Identifier 10.1109/TRO.2022.3226144

simulations [22], [23]. In addition, current solutions rely on fundamental assumptions, such as instantaneous actions without preconditions, which do not hold in real robotic situations.

To tackle these limitations of active inference and to address the two main open challenges of hierarchical deliberation and continual planning, we propose a hybrid combination of active inference and BTs. We then apply this new idea to mobile manipulation in a dynamic retail environment.

A. *Related Work*

In this section, we mainly focus on related work on *reactive action planning and execution*, a class of methods that exploits reactive plans, which are stored structures that contain the behavior of an agent. To begin with, BTs [3], [24] gathered increasing popularity in robotics to specify reactive behaviors. BTs are *de-facto* replacing finite state machines (FSM) in several state-of-the-art systems, for instance in the successor of the ROS Navigation Stack, Nav2 [25] BTs are graphical representations for action execution. The general advantage of BTs is that they are modular and can be composed into more complex higher level behaviors, without the need to specify how different BTs relate to each other. They are also an intuitive representation that modularizes other architectures, such as FSM and decision trees, with proven robustness and safety properties [3]. These advantages and the structure of BTs make them particularly suitable for the class of dynamic problems we are considering in this work, as explained later on in Section II. However, in classical formulations of BTs, the plan reactivity still comes from hard-coded recovery behaviors. This means that highly reactive BTs are usually big and complex and that adding new robotic skills would require revising a large tree. To partially cope with these problems, the authors in [4] proposed a blended reactive task and action planner which dynamically expands a BT at runtime through back-chaining. The solution can compensate for unanticipated scenarios, but cannot handle partially observable environments and uncertain action outcomes. Conflicts due to contingencies are handled by prioritizing (shifting) subtrees. Safronov et al. [5] extended [4] and showed how to handle uncertainty in the BT formulation as well as planning with nondeterministic outcomes for actions and conditions. Other researchers combined the advantages of BTs with the theoretical guarantees on the performance of planning with the planning domain definition language (PDDL) by representing robot task plans as robust logical-dynamical systems (RLDS) [6]. RLDS results in a more concise problem description with respect to using BTs only, but online reactivity is again limited to the scenarios planned offline. For unforeseen contingencies, replanning would be necessary, which is more resource-demanding than reacting, as shown in the experimental results in [6]. The output of RLDS is equivalent to a BT, yet BTs remain more intuitive to compose and have more support from the community with open-source libraries and design tools.

Goal-oriented action planning (GOAP) [26], instead, focuses on online action planning. This technique is used for nonplayer-characters (NPC) in video games [27]. Goals in GOAP do not contain predetermined plans. Instead, GOAP considers atomic behaviors which contain preconditions and effects. The general behavior of an agent can then be specified through very simple FSMs because the transition logic is separated from the states themselves. GOAP generates a plan at run-time by searching in the space of available actions for a sequence that will bring the agent from the starting state to the goal state. However, GOAP requires hand-designed heuristics that are scenario-specific and it is computationally expensive for long-term tasks.

Hierarchical task Planning in the Now (HPN) [28] is an alternate plan and execution algorithm, where a plan is generated backward starting from the desired goal, using A*. HPN recursively executes actions and replans. To cope with the stochasticity of the real world, HPN has been extended to belief HPN (BHPN) [29]. A follow-up work [30] focused on the reduction of the computational burden by implementing *selective replanning* to repair local poor choices or exploit new opportunities without the need to recompute the whole plan which is very costly since the search process is exponential in the length of the plan.

A different method for generating plans in a top-down manner is the hierarchical task network (HTN) [31]. At each step, a high-level task is refined into lower level tasks. In practice [32], the planner exploits a set of standard operating procedures for accomplishing a given task. The planner decomposes the given task by choosing among the available ones until the chosen primitive is directly applicable to the current state. Tasks are then iteratively replaced with new task networks. An important remark is that reactions to failures, time-outs, and external events are still a challenge for HTN planners. HTN requires the designer to write and debug potentially complex domain-specific recipes and, in very dynamic situations, replanning might occur too often.

Finally, active inference is a normative principle underwriting perception, action, planning, decision-making, and learning in biological or artificial agents. Active inference on discrete state spaces [33] is a promising approach to solving the exploitation–exploration dilemma and empowers agents with continuous deliberation capabilities. The application of this theory, however, is still in an early stage for discrete decision-making, where current works only focus on simplified simulations as proof of concept [20], [21], [22], [23]. In [23], for instance, the authors simulated an artificial agent which had to learn and solve a maze given a set of simple possible actions to move (*up, down, left, right, stay*). Actions were assumed instantaneous and always executable. In general, current discrete active inference solutions lack a systematic and task-independent way of specifying prior preferences, which is fundamental to achieving a meaningful behavior, and they never consider action preconditions that are crucial in real-world robotics. As a consequence, plans with conflicting actions which might arise in dynamic environments are never addressed in the current state-of-the-art.

B. *Contributions*

In this work, we propose the hybrid combination of BTs and active inference to obtain more reactive actors with hierarchical deliberation and continual online planning capabilities. We introduce a method to include action preconditions and conflict

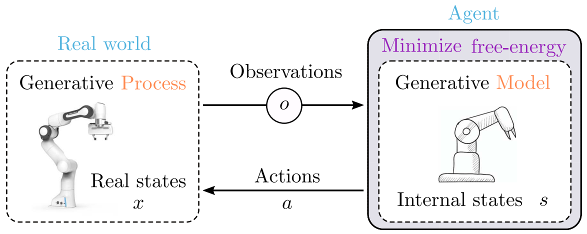


Fig. 1. High-level visualization of an active inference agent. The generative process describes the true causes of the agent's observations o. An agent can apply actions to change the state of the world and to get observations that are aligned with its internal preferences.

resolution in active inference, as well as a systematic way of providing prior preferences through BTs. The proposed hybrid scheme leverages the advantages of online action selection with active inference and it removes the need for complex predefined fallback behaviors while providing convergence guarantees. In addition, we provide extensive mathematical derivations, examples, and code, to understand, test, and reproduce our findings.

C. Article Structure

The rest of this article is organized as follows. In Section II we provide an extensive background on active inference and BTs. Our novel hybrid approach is presented in Section III, and its properties in terms of robustness and stability are analyzed in Section IV. In Section V, we report the experimental evaluation, showing how our solution can be used with different mobile manipulators for different tasks in a retail environment. Section VI contains the discussion. Finally, Section VII concludes this article.

II. Background on Active Inference and BTs

A. Background on Active Inference

Active inference provides a unifying theory for perception, action, decision-making, and learning in biological or artificial agents [33]. Our active inference agent rests on the tuple $(\mathcal{O}, \mathcal{S}, \mathcal{A}, P, Q)$. This is composed of: a finite set of observations $\mathcal{O}$, a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a generative model P, and an approximate posterior Q.

Active inference proposes a solution for action and perception by assuming that actions will fulfill predictions that are based on inferred states of the world, given some observations. The generative model contains beliefs about future states and action plans, where plans that lead to preferred observations are more likely. Perception and action are achieved through the optimization of two complementary objective functions, the variational free-energy F, and the expected free-energy G. These quantities to optimize are derived based on the generative model and the approximate posterior, as detailed later. Variational free-energy measures the fit between the generative model and past and current sensory observations, while expected free-energy scores future possible courses of action according to prior preferences and predicted observations. Fig. 1 depicts the general high level idea. For a first-time reader of active inference, we advise consulting [20] for a more extensive introduction.

In the following, we explain the form of the generative model and how the model parameters relate to states, actions, and observations. Based on this model, we present the expressions for the free-energy and expected free-energy that are used to derive the equations for perception and decision-making. We complement the theory with *pen-and-paper* examples and associated Python code.[1] All the other necessary mathematical derivations are added in the Appendixes.

1) Generative Model: In active inference [20], the generative model P is chosen to be a Markov process that allows to infer the states of the environment and to predict the effects of actions as well as future observations. This is expressed as a joint probability distribution $P(\bar{o}, \bar{s}, \boldsymbol{\eta}, \pi)$, where $\bar{o}$ is a sequence of observations, $\bar{s}$ is a sequence of states, $\boldsymbol{\eta}$ represents model parameters, and π is a plan. In particular, once given fixed model parameters $\boldsymbol{\eta}$ for a task[2], we can write

$$P(\bar{o}, \bar{s}, \pi) = P(\pi) \prod_{\tau=1}^{T} P(s_\tau | s_{\tau-1}, \pi) P(o_\tau | s_\tau). \qquad (1)$$

The full derivation of the generative model in (1), the assumptions under its factorization, and how this joint probability is used to define the free-energy F can be found in Appendix A. The probability distributions in (1) are represented internally by an active inference agent through the following parameters

1) $\boldsymbol{A} \in [0,1]^{r,m}$ is a matrix representation of the conditional probability $P(o_\tau | s_\tau)$, where r is the number of possible observations and m the number of possible states. $\boldsymbol{A}$ is also called the likelihood matrix, and it indicates the probability of observations given a specific state. Each column of $\boldsymbol{A}$ is a categorical distribution. It holds that $P(o_\tau | s_\tau, \boldsymbol{A}) = Cat(\boldsymbol{A} s_\tau)$. For a generic entry $\boldsymbol{A}_{ij} = P(o_\tau = i | s_\tau = j)$.
2) $\boldsymbol{B} \in [0,1]^{m,m}$ represents a transition matrix. In particular $P(s_{\tau+1} | s_\tau,\ a_\tau) = Cat(\boldsymbol{B}_{a_\tau} s_\tau)$. For a symbolic action a_τ in a plan π, $\boldsymbol{B}_{a_\tau}$ represents the probability of state $s_{\tau+1}$ while applying action a_τ from state s_τ. The columns of $\boldsymbol{B}_{a_\tau}$ are categorical distributions.
3) π is a sequence of actions over a time horizon T. $\boldsymbol{\pi}$ is the posterior distribution, a vector holding the probability of different plans. These probabilities depend on the expected free-energy in future time steps under plans given the current belief: $P(\pi) = \sigma(-G(\pi))$. Here, σ indicates the softmax function used to normalize probabilities.

An active inference agent contains also a model $\boldsymbol{D} \in [0,1]^m$ that represents the belief about the initial state at $\tau = 1$. So $P(s_0) = Cat(\boldsymbol{D})$. In addition, an agent also represents prior preferences about desired observations for goal-directed behavior in $\boldsymbol{C} \in \mathbb{R}^r$, such that $P(o_\tau) = \boldsymbol{C}$.

In the context of this article, we do not consider additional generative model parameters, such as the $\boldsymbol{E}$ vector to encode priors over plans used to represent habits [34], [35]. The parameter $\boldsymbol{E}$ could be used to include common sense knowledge in

[1] [Online]. Available: https://github.com/cpezzato/discrete_active_inference/blob/main/discrete_ai/scripts/paper_examples.py

[2] Note that, in the general case, model parameters are not fixed and can be updated as well through active inference [20].

TABLE I
NOTATION FOR ACTIVE INFERENCE

Symbol	Description
$s_\tau \in \{0,1\}^m$	One-hot encoding of the hidden state at time τ, where m is the number of mutually exclusive states in the discrete state space.
$\boldsymbol{s}^\pi_\tau \in [0,1]^m$	Posterior distribution over the state under a plan π, where the elements sum up to one.
$o_\tau \in \{0,1\}^r$	Observation at time τ that can have r mutually exclusive possible values.
$\boldsymbol{o}^\pi_\tau \in [0,1]^r$	Posterior distribution of observations under a plan.
π	Plan specifying a sequence of symbolic actions $\pi = [a_\tau,\ a_{\tau+1}, ..., a_T]^\top$, where T is the time horizon.
$\boldsymbol{\pi} \in [0,1]^p$	Posterior distribution over plans, where p is the number of different plans.
$F(\pi) \in \mathbb{R}$	Plan specific variational free-energy.
$\boldsymbol{F}_\pi \in \mathbb{R}^p$	$\boldsymbol{F}_\pi = (F(\pi_1), F(\pi_2), ...)^\top$ is a column vector containing the free-energy for every plan.
$G(\pi, \tau) \in \mathbb{R}$	Expected free-energy for a plan at time τ.
$\boldsymbol{G}_\pi \in \mathbb{R}^p$	$\boldsymbol{G}_\pi = (G(\pi_1), G(\pi_2), ...)^\top$ is a column vector containing the expected free-energy for every plan.
$\boldsymbol{A} \in [0,1]^{r\times m}$	Likelihood matrix, mapping from hidden states to observations $P(o_\tau \mid s_\tau, \boldsymbol{A}) = Cat(\boldsymbol{A}s_\tau)$.
$\boldsymbol{B}_{a_\tau} \in [0,1]^{m\times m}$	Transition matrix, $P(s_{\tau+1} \mid s_\tau,\ a_\tau) = Cat(\boldsymbol{B}_{a_\tau} s_\tau)$.
$\boldsymbol{C} \in \mathbb{R}^r$	Prior preferences over observations $P(o_\tau) = \boldsymbol{C}$.
$\boldsymbol{D} \in [0,1]^m$	Prior over initial states $P(s_0) = Cat(\boldsymbol{D})$.
σ	Softmax function.

the decision-making process, but this will be addressed in future work. Table I summarizes the notation adopted in this article. The top part contains quantities computed by active inference, while the bottom part the domain parameters required. As explained in the following, these internal models will be used by an active inference agent to compute the free-energy and the expected free-energy.

2) Variational Free-Energy: Given the generative model as before, one can derive an expression for the variational free-energy. By minimizing F, an agent can determine the most likely hidden states given sensory information. The expression for F is given by

$$F(\pi) = \sum_{\tau=1}^{T} \boldsymbol{s}^{\pi\top}_\tau \left[\ln \boldsymbol{s}^\pi_\tau - \ln \left(\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}^\pi_{\tau-1}\right) - \ln \left(\boldsymbol{A}^\top \boldsymbol{o}_\tau\right) \right] \tag{2}$$

where $F(\pi)$ is a plan specific free-energy. The logarithm is considered elementwise. For the derivations please see Appendix B.

3) Perception: According to active inference, both perception and decision-making are based on the minimization of free-energy. In particular, for state estimation, we take partial derivatives of F with respect to the states and set the gradient to zero. The posterior distribution of the state, conditioned by a plan, is given by

$$\boldsymbol{s}^\pi_{\tau=1} = \sigma(\ln \boldsymbol{D} + \ln (\boldsymbol{B}^\top_{a_\tau} \boldsymbol{s}^\pi_{\tau+1}) + \ln (\boldsymbol{A}^\top \boldsymbol{o}_\tau)) \tag{3a}$$

$$\boldsymbol{s}^\pi_{1<\tau<T} = \sigma(\ln (\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}^\pi_{\tau-1}) + \ln (\boldsymbol{B}^\top_{a_\tau} \boldsymbol{s}^\pi_{\tau+1}) + \ln (\boldsymbol{A}^\top \boldsymbol{o}_\tau)) \tag{3b}$$

$$\boldsymbol{s}^\pi_{\tau=T} = \sigma(\ln (\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}^\pi_{\tau-1}) + \ln (\boldsymbol{A}^\top \boldsymbol{o}_\tau)) \tag{3c}$$

where σ is the softmax function. The column of $\boldsymbol{B}^\top_{a_\tau}$ are normalized. For the complete derivation, please see to Appendix C. Note that when $\tau = 1$ it holds $\ln (\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}^\pi_{\tau-1}) = \ln \boldsymbol{D}$. We provide below an example of state update and highlight the effect of uncertain action outcomes in the state estimation process.

Example 1. State Estimation: An active inference agent lives in a simple world composed of one state which can have two possible values. The only action the agent can do is to stay still ($\boldsymbol{B}_{idle}$), so $\pi = a_\tau \forall \tau$ with $a_\tau = idle$. However, there are some chances that unwanted transitions between states can occur. The agent can only predict one step ahead ($T = 2$) and it receives an observation $\boldsymbol{o}_{\tau=1}$ at the start, that is related to the state through $\boldsymbol{A}$. The agent has no prior information about the initial state or future observations, thus $\boldsymbol{D}$ is uniform as well as the initial guess about the posterior distributions of the state. This is modeled as follows:

$$\boldsymbol{A} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix},\ \boldsymbol{B}_{idle} = \begin{bmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{bmatrix},\ \boldsymbol{D} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\boldsymbol{o}_{\tau=1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ \boldsymbol{o}_{\tau=2} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},\ \boldsymbol{s}^\pi_{\tau=1} = \boldsymbol{s}^\pi_{\tau=2} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

The update of the posterior distribution for state estimation is, according to (3)

$$\boldsymbol{s}^\pi_{\tau=1} = \sigma\left(\ln \begin{bmatrix} .5 \\ .5 \end{bmatrix} + \ln \left(\begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix} \begin{bmatrix} .5 \\ .5 \end{bmatrix} \right) + \ln \left(\begin{bmatrix} .9 & .1 \\ .1 & .9 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \right) = \begin{bmatrix} .9 \\ .1 \end{bmatrix}$$

$$\boldsymbol{s}^\pi_{\tau=2} = \sigma\left(\ln \left(\begin{bmatrix} .8 & .2 \\ .2 & .8 \end{bmatrix} \begin{bmatrix} .9 \\ .1 \end{bmatrix} \right) + \ln \left(\begin{bmatrix} .9 & .1 \\ .1 & .9 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \right) = \begin{bmatrix} .74 \\ .26 \end{bmatrix}.$$

As commonly done in active inference literature, a small number (for instance e^{-16} [34]) is added when computing the logarithms. This prevents numerical errors in case of $\ln(0)$. Note that, if there would be no uncertainty on the actions the agent can take, i.e., $\boldsymbol{B}_{idle}$ is the identity in this case, then the estimated hidden state at $\tau = 2$ would be $\boldsymbol{s}^\pi_{\tau=2} = [0.9, 0.1]^\top$. The agent would then be more confident under this action.

4) Expected Free-Energy: Active inference unifies action selection and perception by assuming that actions fulfill predictions based on inferred states. Since the internal model can be biased toward preferred states or observations (*prior desires*), active inference induces actions that will bring the current beliefs toward the preferred states. An agent builds beliefs about future states which are then used to compute the expected free-energy. The latter is necessary to evaluate alternative plans. Plans that lead to preferred observations are more likely. Preferred observations are specified in the model parameter $\boldsymbol{C}$. This enables action to realize the next (proximal) *observation* predicted by the plan that leads to (distal) goals. The expected free-energy for a plan π at time τ is given by

$$G(\pi,\tau) = \underbrace{\boldsymbol{o}_\tau^{\pi\top}\left[\ln \boldsymbol{o}_\tau^{\pi} - \ln \boldsymbol{C}\right]}_{\text{Reward seeking}} \underbrace{-\text{diag}(\boldsymbol{A}^\top \ln \boldsymbol{A})^\top \boldsymbol{s}_\tau^{\pi}}_{\text{Information seeking}}. \tag{4}$$

The diag() function simply takes the diagonal elements of a matrix and puts them in a column vector. This is just a method to extract the correct matrix entries in order to compute the expected free-energy [34]. By minimizing expected free-energy the agent balances reward and information seeking (see Appendix D for derivations). Reward and information-seeking behaviors that arise from the formulation of the expected free-energy are illustrated, respectively, in Examples 2 and 3. To keep the examples reasonably simple to be computed by hand, we consider that the posterior over states according to different plans have already been computed following (3), and are given.

Example 2. Reward Seeking: Consider an agent has computed the posterior distribution of the states $\boldsymbol{s}_\tau^{\pi_1}$ and $\boldsymbol{s}_\tau^{\pi_2}$ under two different plans π_1 and π_2. The agent has a preference for a particular value of the state encoded in $\boldsymbol{C}$. The model for this example is the following:

$$\boldsymbol{A} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{s}_\tau^{\pi_1} = \begin{bmatrix} 0.95 \\ 0.05 \end{bmatrix}, \boldsymbol{s}_\tau^{\pi_2} = \begin{bmatrix} 0.05 \\ 0.95 \end{bmatrix}.$$

Let us consider the reward-seeking term in (4) (note that the information-seeking term is equal in both plans for this example). The observations expected under the two different plans are

$$\boldsymbol{o}_\tau^{\pi_1} = \boldsymbol{A}\boldsymbol{s}_\tau^{\pi_1} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}, \boldsymbol{o}_\tau^{\pi_2} = \boldsymbol{A}\boldsymbol{s}_\tau^{\pi_2} = \begin{bmatrix} 0.14 \\ 0.86 \end{bmatrix}.$$

Intuitively, according to $\boldsymbol{C}$, the first plan is preferable and it should have the lowest expected free-energy because it leads to preferred observations with higher probability. Numerically

$$\boldsymbol{o}_\tau^{\pi_1\top}\left[\ln \boldsymbol{o}_\tau^{\pi_1} - \ln \boldsymbol{C}\right] = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}^\top$$
$$\times \left[\ln \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix} - \ln \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right] \approx 1.84.$$

Similarly for π_2, $\boldsymbol{o}_\tau^{\pi_2\top}[\ln \boldsymbol{o}_\tau^{\pi_2} - \ln \boldsymbol{C}] \approx 13.35$. As can be noticed, the plan that brings the posterior state closest to the preference specified in $\boldsymbol{C}$ leads to the lowest reward-seeking term.

Example 3. Information Seeking: Let us consider a variation of Example 2. The agent is not given any preference for a specific state, and the likelihood matrix $\boldsymbol{A}$ encodes now the fact that observations are expected to provide more precise information when the agent is in the second state (second column of $\boldsymbol{A}$). This results in the following models:

$$\boldsymbol{A} = \begin{bmatrix} 0.7 & 0.1 \\ 0.3 & 0.9 \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{s}_\tau^{\pi_1} = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}, \boldsymbol{s}_\tau^{\pi_2} = \begin{bmatrix} 0.1 \\ 0.9 \end{bmatrix}.$$

We expect that the plan which leads to a state with less ambiguous information has the lowest information-seeking term. For the first plan

$$-\,diag(\boldsymbol{A}^\top \ln \boldsymbol{A})^\top \boldsymbol{s}_\tau^{\pi_1} =$$
$$-\,diag\left(\begin{bmatrix} 0.7 & 0.3 \\ 0.1 & 0.9 \end{bmatrix} \ln \begin{bmatrix} 0.7 & 0.1 \\ 0.3 & 0.9 \end{bmatrix}\right)^\top \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} \approx 0.58.$$

For the second plan, the information-seeking term is instead ≈ 0.35. The state achieved with the second plan generates less ambiguous observations. Plans that lead to the lowest ambiguity in sensory information, and thus, minimize G, are preferred.

In more complex examples, minimizing G leads to a balance in reward and information seeking. For a fully fledged exploration–exploitation problem, see a recently released Python library for active inference [36] which contains an interactive and visual example[3] of the emergent behavior of a rat in a grid world which has to collect cues to disclose the location of the reward.

a) Planning and decision-making: Taking the gradient of F with respect to plans, and recalling that the generative model specifies the approximate posterior over plans as a softmax function of the expected free-energy [33] it holds that

$$\boldsymbol{\pi} = \sigma(-\boldsymbol{G}_\pi - \boldsymbol{F}_\pi) \tag{5}$$

where the vector $\boldsymbol{\pi}$ encodes the posterior distribution over plans reflecting the predicted value of each plan. $\boldsymbol{F}_\pi = (F(\pi_1), F(\pi_2), \ldots)^\top$ and $\boldsymbol{G}_\pi = (G(\pi_1), G(\pi_2), \ldots)^\top$. See Appendix E for the details.

b) Plan independent state-estimation: Given the probability over p possible plans, and the plan dependent states $\boldsymbol{s}_\tau^\pi$, we can compute the overall probability distribution for the states over time through the Bayesian model average

$$\boldsymbol{s}_\tau = \sum_i \boldsymbol{s}_\tau^{\pi_i} \boldsymbol{\pi}_i, \text{ where } i \in \{1, \ldots, p\} \tag{6}$$

where $\boldsymbol{s}_\tau^{\pi_i}$ is the probability of a state at time τ under plan i and $\boldsymbol{\pi}_i$ is the probability of plan i. This is the average prediction for the state at a certain time, so $\boldsymbol{s}_\tau$, according to the probability of each plan. In other words, this is a weighted average over different models. Models with high probability receive more weight, while models with lower probabilities are discounted.

c) Action selection: The action for the agent to be executed is the first action of the most likely plan

$$\lambda = \max(\underbrace{[\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_p]}_{\boldsymbol{\pi}^\top}), \; a_\tau = \pi_\lambda(\tau = 1) \tag{7}$$

where λ is the index of the most likely plan.

Example 4: Plan and Action Selection: By computing the expected free-energy of Exercise 2 using (4) and including the information seeking term, we obtain $\boldsymbol{G}_\pi = [G(\pi_1), G(\pi_2)]^\top \approx [2.16, 13.68]^\top$. For the sake of this example, let us assume the free-energy is equal for both plans, that is $\boldsymbol{F}_\pi = [F(\pi_1), F(\pi_2)]^\top \approx [1.83, 1.83]^\top$. The posterior distribution over plans is then

$$\boldsymbol{\pi} = \sigma\left(-\begin{bmatrix} 2.16 \\ 13.68 \end{bmatrix} - \begin{bmatrix} 1.83 \\ 1.83 \end{bmatrix}\right) \approx \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}.$$

As can be seen, the most likely plan is the first one, in accordance with the conclusions of Exercise 2. The action to be applied by the agent is the first action of π_1.

[3] [Online]. Available: https://pymdp-rtd.readthedocs.io/en/latest/notebooks/cue_chaining_demo.html

Algorithm 1: Action Selection With Active Inference

```
Set $\boldsymbol{C}$                                   ▷ prior preferences
for $\tau = 1 : T$ do
    If not specified, get state from $\boldsymbol{D}$ if $\tau == 1$
    If not specified, get observation from $\boldsymbol{A}$
    Compute $F$ for each plan                          ▷ (2)
    Update posterior state $\boldsymbol{s}_\tau^\pi$   ▷ (3)
    Compute $G$ for each plan                          ▷ (4)
    Bayesian model averaging                           ▷ (6)
    Action selection                                   ▷ (7)
end for
Return $a$                                             ▷ Preferred action
```

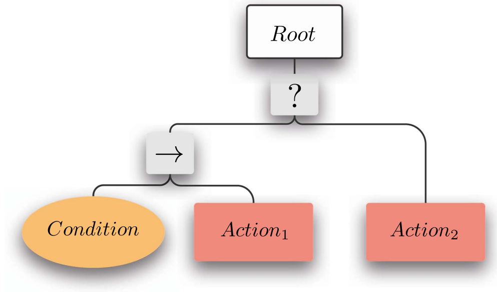


Fig. 2. Example of BT. The *tick* traverses the tree starting from the *Root*. If $Condition$ is true $Action_1$ is executed. Then, if $Action_1$ returns success, the *Root* returns success, otherwise $Action_2$ is executed.

The active inference algorithm is summarized in the pseudocode in Algorithm 1.

5) Multiple Sets of States and Observations: The active inference model introduced in this section can also handle multiple sets of independent states and observations [34]. A famous example in the active inference literature covers a rat in a T-maze [20]. The rat is seeking a reward (cheese) but the location of the cheese is only known after receiving a cue. In this case, one set of states encodes the location of the rat, while another set encodes the initially unknown location of the cheese. Each independent set of states is called a *state factor*. In the same way, there can be multiple sets of observations coming from different sensors. Each set is a different *observation modality*. The authors in [34] and [35] provided a step-by-step tutorial on active inference with fully worked out toy examples including multiple factors and modalities. We provide explicit models for our robotic case with multiple state factors and observations in Section III-A.

B. Background on BTs

We now describe the high-level concepts at the basis of BTs according to previous work, such as [3], [24]. These concepts will be useful to understand the novel hybrid scheme proposed in the next section. A BT is a directed tree composed of nodes and edges that can be seen as a graphical modeling language. It provides a structured representation for the execution of actions that are based on conditions and observations in a system. The nodes in a BT follow the classical definition of parents and children. The root node is the only node without a parent, while the leaf nodes are all the nodes without children. In a BT, the nodes can be divided into control flow nodes (*Fallback*, *Sequence*, *Parallel*, or *Decorator*), and into execution nodes (*Action* or *Condition*) which are the leaf nodes of the tree. When executing a given BT in a control loop, the root node sends a *tick* to its child. A tick is nothing more than a signal that allows the execution of a child. The tick propagates in the tree following the rules dictated by each control node. A node returns a status to the parent, which can be *running* if its execution has not finished yet, *success* if the goal is achieved, or *failure* in the other cases. At this point, the return status is propagated back up the tree, which is traversed again following the same rules. The most important control nodes are as follows.

Fallback Nodes: A fallback node ticks its children from left to right. It returns *success* (or *running*) as soon as one of the children returns *success* (or *running*). When a child returns *success* or *running*, the fallback does not tick the next child, if present. If all the children return *failure*, the fallback returns *failure*. This node is graphically identified by a gray box with a question mark "?."

Sequence Nodes: The sequence node ticks its children from left to right. It returns *running* (or *failure*) as soon as a child returns *running* (or *failure*). The sequence returns *success* only if all the children return *success*. If a child returns *running* or *failure*, the sequence does not tick the next child, if present. In the library, we used to implement our BTs [37] the *sequence* node, indicated with [→], keeps ticking a running child, and it restarts only if a child fails. Faconti [37] also provides *reactive sequences* [$\rightarrow^R$], where every time a sequence is ticked, the entire sequence is restarted from the first child.

The execution nodes are *Actions* and *Conditions*.

Action Nodes: An Action node performs an action in the environment. While an action is being executed, this node returns *running*. If the action is completed correctly it returns *success*, while if the action cannot be completed it returns *failure*. Actions are represented as red rectangles.

Condition Nodes: A Condition node determines if a condition is met or not, returning *success* or *failure* accordingly. Conditions never return *running* and do not change any states or variables. They are represented as orange ovals.

An example BT is given in Fig. 2.

III. Active Inference and BTs for Reactive Action Planning and Execution

In this section, we introduce our novel approach using BTs and active inference. Even though active inference is a very promising theory, from a computational perspective computing the expected free-energy for each possible plan that a robot might take is cost-prohibitive. This curse of dimensionality is due to the combinatorial explosion when looking deep into the future [33]. To solve this problem, we propose to replace *deep plans* with *shallow decision trees* that are *hierarchically composable*. This will allow us to simplify our offline plans, exploit opportunities, and act intelligently to resolve local unforeseen contingencies. Our idea consists of two main intuitions as follows.

1) To avoid combinatorial explosion while planning and acting with active inference for long-term tasks we specify the nominal behavior of an agent through a BT, used as a prior. In doing so, BTs provide *global reactivity to foreseen situations*.
2) To avoid coding every possible contingency in the BT, we program only desired states offline, and we leave action selection to the online active inference scheme. Active inference provides then *local reactivity to unforeseen situations*.

To achieve such a hybrid integration, and to be able to deploy this architecture on real robotic platforms, we addressed the following three fundamental problems: 1) How to define the generative models for active inference in robotics, 2) how to use BTs to provide priors as desired states to active inference, and 3) how to handle action preconditions in active inference, and possible conflicts which might arise at run-time in a dynamic environment.

A. *Definition of the Models for Active Inference*

The world in which a robot operates needs to be abstracted such that the active inference agent can perform its reasoning processes. In this work, we operate in a continuous environment with the ability of sensing and acting through symbolic decision-making. In the general case, the decision-making problem will include multiple sets of states, observations, and actions. Each independent set of states is a factor, for a total of n_f factors. For a generic factor f_j, where $j \in \mathcal{J} = \{1, \ldots, n_f\}$, the corresponding state factor is

$$s^{(f_j)} = \left[s^{(f_j,1)}, s^{(f_j,2)}, \ldots, s^{(f_j,m^{(f_j)})}\right]^\top$$
$$\mathcal{S} = \left\{s^{(f_j)} | j \in \mathcal{J}\right\} \tag{8}$$

where $m^{(f_j)}$ is the number of mutually exclusive symbolic values that a state factor can have. Each entry of $s^{(f_j)}$ is a real value between 0 and 1, and the sum of the entries is 1. This represents the current belief state. Then, we define $x \in \mathcal{X}$ the continuous states of the world and the internal states of the robot are accessible through the symbolic perception system. The role of this perception system is to compute the symbolic observations based on the continuous state x, such that they can be manipulated by the discrete active inference agent. Observations o are used to build a probabilistic belief about the current state. Assuming one set of observations per state factor with $r^{(f_j)}$ possible values, it holds

$$o^{(f_j)} = \left[o^{(f_j,1)}, o^{(f_j,2)}, \ldots, o^{(f_j,r^{(f_j)})}\right]^\top$$
$$\mathcal{O} = \left\{o^{(f_j)} | j \in \mathcal{J}\right\}. \tag{9}$$

In addition, the robot has a set of symbolic skills to modify the corresponding state factor

$$a_\tau \in \alpha^{(f_j)} = \left\{a^{(f_j,1)}, a^{(f_j,2)}, \ldots, a^{(f_j,k^{(f_j)})}\right\}$$
$$\mathcal{A} = \left\{\alpha^{(f_j)} | j \in \mathcal{J}\right\} \tag{10}$$

where $k^{(f_j)}$ is the number of actions that can affect a specific state factor f_j. Each generic action $a^{(f_j,\cdot)}$ has associated a symbolic name, *parameters*, *pre-* and *postconditions*

Action $a^{(f_j,\cdot)}$	**Preconditions**	**Postconditions**
$\texttt{action\_name}(par)$	$\texttt{prec}_{a^{(f_j,\cdot)}}$	$\texttt{post}_{a^{(f_j,\cdot)}}$

where $\texttt{prec}_{a^{(f_j,\cdot)}}$ and $\texttt{post}_{a^{(f_j,\cdot)}}$ are *first-order logic predicates* that can be evaluated at run-time. A logical predicate is a Boolean-valued function $\mathcal{P} : \mathcal{X} \rightarrow \{\texttt{true}, \texttt{false}\}$.

Finally, we define the logical state $l^{(f_j)}$ as a one-hot encoding of $s^{(f_j)}$. We indicate as $\mathcal{L}_c(\tau) = \{l^{(f_j)} | j \in \mathcal{J}\}$ the (time varying) *current* logical state of the world. Defining a logic state based on the probabilistic belief s built with active inference, instead of directly using the observation of the states o, increases robustness against noisy sensor readings, as we will explain in Example 12. Given the model of the world just introduced, we can now define for each factor f_j the likelihood matrix $\boldsymbol{A}^{(f_j)}$, the $k^{(f_j)}$ transition matrices $\boldsymbol{B}_{a_\tau^{(f_j,\cdot)}}$, and the prior preferences $\boldsymbol{C}^{(f_j)}$. When an observation is available, $\boldsymbol{A}^{(f_j)}$ provides information about the corresponding value of a state factor $s^{(f_j)}$. For a particular state, the probability of a state observation pair $o_\tau^{(f_j)}$ and $s_\tau^{(f_j)}$ is given by $\boldsymbol{A}^{(f_j)} \in \mathbb{R}^{r^{(f_j)} \times m^{(f_j)}}$. In case a state factor is observable with full certainty, each state maps into the corresponding observation thus the likelihood matrix is the identity of size $m^{(f_j)}$, $I_{m^{(f_j)}}$. Note that knowing the mapping between observations and states does not necessarily mean that we can observe all the states at all times. Observations can be present or not, and when they are the likelihood matrix indicates the relation between that observation and the state. This relation can be more complex and incorporate uncertainty in the mapping as well. To define the transition matrices, we need to encode in a matrix form the effects of each action on the relevant state factors. The probability of a ending up in a state $s_{\tau+1}^{(f_j)}$, given $s_\tau^{(f_j)}$ and action $a_\tau^{(f_j,\cdot)}$ is given by

$$P(s_{\tau+1}^{(f_j)} | s_\tau^{(f_j)}, a_\tau^{(f_j,\cdot)}) = Cat(\boldsymbol{B}_{a_\tau^{(f_j,\cdot)}} s_\tau^{(f_j)})$$
$$\boldsymbol{B}_{a_\tau^{(f_j,\cdot)}} \in \mathbb{R}^{m^{(f_j)} \times m^{(f_j)}}. \tag{11}$$

In other words, we define $\boldsymbol{B}_{a_\tau^{(f_j,\cdot)}}$ as a square matrix encoding the postconditions of action $a_\tau^{(f_j,\cdot)}$. The prior preferences over observations (or states) need to be encoded, for each factor, in $\boldsymbol{C}^{(f_j)} \in \mathbb{R}^{m^{(f_j)}}$, with $\mathcal{C} = \{\boldsymbol{C}^{(f_j)} | j \in \mathcal{J}\}$. The higher the value of a preference, the more preferred a particular state is, and vice-versa. Priors are formed according to specific desires and they will be used to interface active inference and BTs. Finally, one has also to define the vector encoding the initial belief about the probability distribution of the states, that is $\boldsymbol{D}^{(f_j)} \in \mathbb{R}^{m^{(f_j)}}$. This vector is normalized, and when no prior information is available, each entry will be $1/m^{(f_j)}$. In this work, we assume the model parameters, such as likelihood and transition matrices to be known. However, one could use the free-energy minimization to learn them [38].

Example 5: Consider a mobile manipulator in a retail environment. We want the robot to be able to decide when to navigate to a certain goal location. To achieve so, we need one state factor $s^{(loc)}$, one observation $o^{(loc)}$, and one symbolic action $a^{(loc,1)} =$ `moveTo(goal)`. The robot can also decide not to do anything, so $a^{(loc,2)} =$ `idle`.

Actions	Preconditions	Postconditions
`moveTo(goal)`	-	$l^{(loc)} = [1\ 0]^\top$
`idle`	-	-

The current position in space of the robot is a continuous value $x \in \mathcal{X}$. However, during execution the robot is given an observation $o^{(loc)}$ which indicates simply if the `goal` has been reached or not. In this case, $\boldsymbol{A}^{(loc)} = I_2$. The agent is then constantly building a probabilistic belief $s^{(loc)}$ encoding the chances of being at the `goal`. In case the robot has not yet reached the goal, a possible configuration at time τ is the following:

$$o^{(loc)} = \begin{bmatrix} \texttt{isAt(goal)} \\ \texttt{!isAt(goal)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; s^{(loc)} = \begin{bmatrix} 0.08 \\ 0.92 \end{bmatrix} \tag{12}$$

$$l^{(loc)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; \boldsymbol{B}_{moveTo} = \begin{bmatrix} 0.95 & 0.9 \\ 0.05 & 0.1 \end{bmatrix}, \; \boldsymbol{B}_{idle} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The preference over a state to be reached is given through $\boldsymbol{C}^{(loc)}$. A robot wanting to reach a location will have, for instance, a preference $\boldsymbol{C}^{(loc)} = [1, 0]^\top$.

The transition matrix $\boldsymbol{B}_{\text{moveTo}}$ encodes the probability of reaching a goal location through the action `moveTo(goal)`, which might fail with a certain probability. We also encode an `Idle` action, which does not modify any state, but it provides information on the outcome of the action selection process as we will see in the next subsections. In this simple case, the world state is just a single state factor $s^{(\text{loc})}$. On the other hand, in later more complicated examples, the world state will contain all the different aspects of the world, for which a probabilistic representation of their value is built and updated.

Using the proposed problem formulation for the active inference models, we can abstract redundant information which is not necessary to make high-level decisions. For instance, in the example above, we are not interested in building a probabilistic belief of the current robot position. To decide if to use the action `moveTo(goal)` or not, it is sufficient to encode if the `goal` has been reached or not.

B. BTs Integration: Planning Preferences, Not Actions

To achieve a meaningful behavior in the environment through active inference, we need to encode specific desires into the agent's brain through $\mathcal{C} = \{\boldsymbol{C}^{(f_j)} | j \in \mathcal{J}\}$.

A Prior as BT: We propose to extend the available BT nodes in order to be able to specify desired states to be achieved as leaf nodes. We introduce a new type of leaf node called *prior nodes*, indicated with a green hexagon. These nodes can be seen as standard action nodes but instead of commanding an action, they simply set the desired value of a state in $\mathcal{C}$ and they run active inference for action selection. The prior node is then just a leaf node in the BT which returns: `Success` if a state is achieved, `Running` while trying to achieve it, or `Failure` if for some reason it is not possible to reach the desired state. The return statuses are according to the outcome of our reactive action selection process, as explained in Section III-D.

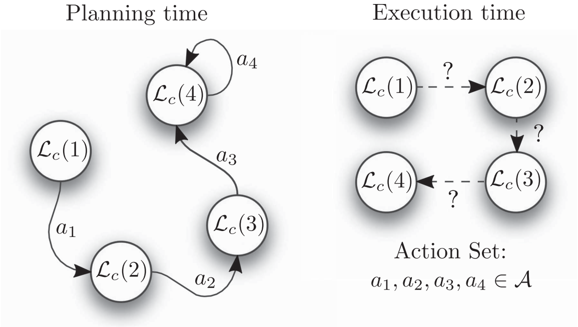


Fig. 3. Path among states is planned offline using the available set of actions but only the sequence of states is provided at run-time. Actions are chosen online from the available set with active inference.

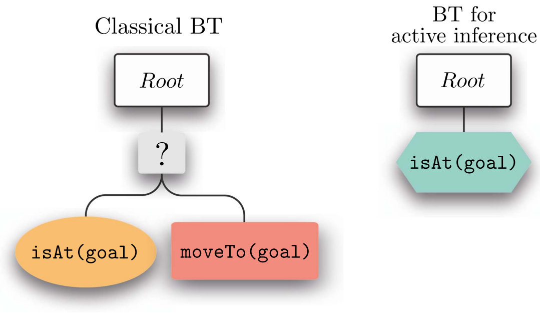


Fig. 4. BT to navigate to a location using a classical BT and a BT for active inference. One action `moveTo(goal)` is available and one condition `isAt(goal)` provides information if the current location is at the goal. The prior node for active inference (green hexagon) sets the desired prior and runs the action selection process.

Subgoals through BTs: To reach a distal goal state, we plan achievable subgoals in the form of desired logical states l, according to the available actions that a robot possesses. This idea of using subgoals was already used in [23], but in our solution with BTs, we provide a task-independent way to define subgoals which is only based on the set of available skills of the robot, such that we can make sure that it can complete the task. At planning time, we define the ideal sequence of states and actions to complete a task such that subsequent subgoals (or logical desired states) are achievable by means of one action. This can be done manually or through automated planning. At run-time, however, we only provide the sequence of states to the algorithm, as in Fig. 3.

Example 6: To program the behavior of the robot in Example 5 to visit a certain goal location, the BT will set the prior over $s^{(loc)}$ to $\boldsymbol{C}^{(loc)} = [1, 0]^\top$ meaning that the robot would like to sense to be at `goal`.

A classical BT and a BT for active inference with prior nodes are reported in Fig. 4. Note that the action is left out in the BT for active inference because these are selected at runtime. In this particular case, the condition `isAt(goal)` can be seen as the desired observation to obtain.

Note that the amount of knowledge (i.e., number of states and actions) that is necessary to code a classical BT or our active inference version in Example 6 is the same. However, we abstract the fallback by planning in the state space and not in the action space. Instead of programming the action `moveTo(goal)` we only set a prior preference over the state `isAt(goal)` since the important information is retained in the state to be achieved rather than in the sequence of actions to do so. Action selection through active inference will then select the appropriate skills to match the current state of the world with the desired one, minimizing this discrepancy through free-energy minimization.

Example 7: Consider the scenario in Example 5 with the prior as in Example 6. The prior is specifying a preference over being at the desired goal location, but the mobile manipulator is not. The plan generated at runtime with Algorithm 1 (see Examples 2 and 4) would be to perform the action `moveTo(goal)`, since this increases the probability of getting an observation `isAt(goal)`.

As we will thoroughly explain in Section IV-A, our algorithm creates online a stable region of attraction from the current state to the goal state instead of planning it offline through fallbacks. See Example 11 for a concrete case.

C. Action Preconditions and Conflicts

Past work on active inference, such as [23], was based on the assumption that actions were always executable and nonconflicting, but these do not hold in more realistic scenarios.

Action Preconditions in Active Inference: We propose to encode action preconditions as desired logical states that need to hold to be able to execute a particular action. This is illustrated in the next example.

Example 8: We add one more action to the set of skills of our mobile manipulator: `pick(obj)` and the relative transition matrix $\boldsymbol{B}_{pick}$. The action templates are extended as follows:

Actions	**Preconditions**	**Postconditions**
`moveTo(goal)`	-	$l^{(loc)} = [1\ 0]^\top$ $l^{(reach)} = [1\ 0]^\top$
`pick(obj)`	`isReachable(obj)`	$l^{(hold)} = [1\ 0]^\top$

$$o^{(hold)} = \begin{bmatrix} \texttt{isHolding(obj)} \\ \texttt{!isHolding(obj)} \end{bmatrix}, \quad B_{pick} = \begin{bmatrix} 0.95 & 0.9 \\ 0.05 & 0.1 \end{bmatrix}$$

$$o^{(reach)} = \begin{bmatrix} \texttt{isReachable(obj)} \\ \texttt{!isReachable(obj)} \end{bmatrix} \tag{13}$$

where we added a new logical state $l^{(\mathrm{hold})}$, the relative belief $s^{(\mathrm{hold})}$, and observation $o^{(\mathrm{hold})}$, which indicates if the robot is holding the object `obj`. In the simplest case, we suppose that the only precondition for successful grasping is that `obj` is reachable. We then add a logical state $l^{(\mathrm{reach})}$, as well as $s^{(\mathrm{reach})}$ and $o^{(\mathrm{reach})}$, to provide active inference with information about this precondition. $o^{(\mathrm{reach})}$ can be built for instance trying to compute a grasping pose for a given object. The robot can act on the state $l^{(\mathrm{hold})}$ through `pick`, and it can act on $l^{(\mathrm{reach})}$ through `moveTo`.

Conflicts Resolution in Active Inference: Conflict resolution due to dynamic changes in the environment and online decision-making is handled through modification of the prior preferences in $\mathcal{C}$. The BT designed offline specifies at runtime the desired state to be achieved. This is done by populating the prior preference over a state with the value of one. Note that if there is no goal, the preferences over states are set to zero everywhere so there is no incentive to act to achieve a different state. Given some preferences over states, the online decision-making algorithm with active inference selects an action, and checks if the preconditions are holding according to the current belief state. If so, the action is executed, if not, the missing preconditions are added to the current preferred state with a higher preference (i.e., > 1), in our case with value 2. This can lead to conflicts with the original BT [4], that is the robot might want to simultaneously achieve two conflicting states. However, the state relative to a missing precondition has higher priority (i.e., > 1) by construction. As explained in Algorithm 2, if preconditions are missing at runtime, action selection is performed again with the updated prior $\mathcal{C}$, such that actions that will satisfy them are more likely. With our method, there is no need to explicitly detect a conflict and reordering a BT then, as was done in past work on the dynamic expansion of BTs [4]. In fact, since the decision-making with active inference happens continuously during task execution, once a missing precondition is met this is removed from the current desired state. Thus, the only remaining preference is the one imposed by the BT, which can now be resumed. This leads to a natural conflict resolution and plan resuming without *ad hoc* recovery mechanisms. The advantage of active inference is that we can represent *which* state is important but also *when* with different values of preference. Since missing preconditions are added to the current prior with a higher preference with respect to the offline plan, this will induce a behavior that can initially go against the initial BT because the new desire is more appealing to be satisfied. Conflict resolution is then achieved by *locally updating* prior desires about a state, giving them higher preference. The convergence analysis of this approach is reported in Section IV, and a concrete example of conflict resolution in a robotic scenario is presented in Section V-D, Example 13.

D. Complete Control Scheme

Our solution is summarized in Algorithm 2 and Fig. 5. Every time a BT is ticked, given a certain frequency, Algorithm 2 is run. The symbolic perception layer takes the sensory readings and translates these continuous quantities into logical observations. This can be achieved through user-defined models according to the specific environment and sensors available. The logical observations are used to perform belief updating to keep a probabilistic representation of the world in $\mathcal{S}$. Then, the logical state $\mathcal{L}_c(\tau)$ is formed. Every time a *prior* node in the BT is ticked, the corresponding priors in $\mathcal{C}$ are set.

For both missing preconditions and conflicts, high priority priors are removed from the preferences $\mathcal{C}$ whenever the preconditions are satisfied or the conflicts resolved (lines `6-10`), allowing to resume the nominal flow of the BT. Active inference

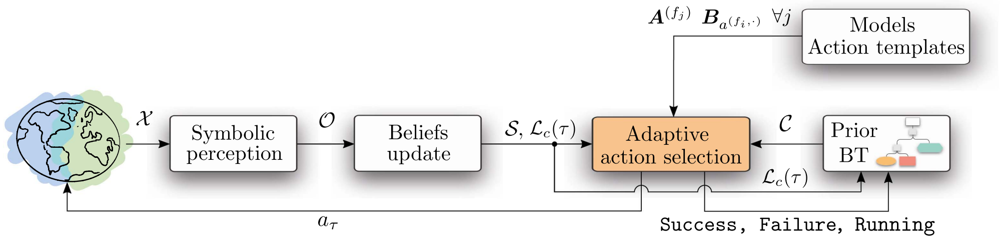


Fig. 5. Overview of the control architecture for reactive action planning and execution using active inference. Adaptive action selection is performed according to Algorithm 2. The symbolic perception module computes the symbolic observations based on the continuous state. In this work we assume this mapping known and encode it through simple rules based on measurements from the robot's sensors. However learning methods to define this relationship from data could be employed as well.

from Algorithm 1 is then run for action selection. If no action is required since the logical state corresponds to the prior, the algorithm returns `Success`, otherwise, the selected action's preconditions are checked and eventually pushed with higher priority. Then, action selection is performed with the updated prior. This procedure is repeated until either an executable action is found, returning `Running`, or no action can be executed, returning `Failure`. The case of `Failure` is handled through the global reactivity provided by the BT. This creates dynamic and stable regions of attraction, as explained in Section IV-A, by means of sequential controller composition [39] (lines `17-31` of Algorithm 2).

Crucially, in this work, we propose the new idea of using *dynamic priors*. For a factor f_j, $\boldsymbol{C}^{(f_j)}$ is not fixed *a priori* as in past active inference works, but instead, it can change over time according to the BT for a task. This allows preconditions checking and conflict resolution within active inference. A robot can follow a long programmed routine while autonomously taking decisions to locally compensate for unexpected events. This reduces considerably the need for hard-coded fallbacks, allowing to compress the BT to a minimal number of nodes.

Algorithm 2: Pseudo-Code for Adaptive Action Selection.

```
 1: Get desired prior and parameters from BT:
 2: $\mathcal{C}$, param ← BT                              ▷ With priority 1
 3: Set current observations, beliefs and logical state:
 4: Set $\mathcal{O}$, $\mathcal{S}$, $\mathcal{L}_c(\tau)$
 5: Remove preferences with high-priority (i.e., $> 1$) if
    satisfied:
 6: for all priors $\boldsymbol{C}^{(f_j)}$ with preference $\geq 1$ do
 7:   if $l_c^{(f_j)}$ holds then
 8:     Remove pushed preference for $l_c^{(f_j)}$;
 9:   end if
10: end for
11: Run active inference given $\mathcal{O}$, $\mathcal{S}$ and $\mathcal{C}$:
12: $a_\tau$ ← Action_selection($\mathcal{O}$,$\mathcal{S}$,$\mathcal{C}$)   ▷ Algorithm 1
13: Update $\mathcal{L}_c(\tau)$
14: if $a_\tau$ == Idle then
15:   return Success;                            ▷ No action required
16: else
17:   Check action preconditions:
18:   while $a_\tau$ !=Idle
19:     if prec$_{a_\tau}$ $\in \mathcal{L}_c(\tau)$ OR prec$_{a_\tau}$ $= \emptyset$ then
20:       Execute($a_\tau$);
21:       break, return Running;                   ▷ Executing $a_\tau$
22:     else
23:       Push missing preconditions in $\mathcal{C}$:
24:       $\mathcal{C}$ ← prec$_{a_\tau}$;                        ▷ With priority 2
25:       Exclude $a_\tau$ and re-run Algorithm 1:
26:       Remove($a_\tau$);
27:       $a_\tau$ ← Action_selection($\mathcal{O}$,$\mathcal{S}$,$\mathcal{C}$)
28:       if $a_\tau$ == Idle then
29:         return Failure;                          ▷ No solution
30:       end if
31:     end if
32:   end while
33: end if
```

IV. Theoretical Analysis

A. Analysis of Convergence

We provide a theoretical analysis of the proposed control architecture. There are two possible scenarios that might occur at run-time. Specifically, the environment might or might not differ from what has been planned offline through BTs. These two cases are analyzed in the following to study the convergence to the desired goal of our proposed solution.

1) Dynamic Environment is as Planned: In a nominal execution, where the environment in which a robot is operating is the same as the one at planning time, there is a one-to-one equivalence between our approach and a classical BT formulation. This follows directly by the fact that the BT is defined according to Section III-B, so each subsequent state is achievable by means of one single action. At any point of the task, the robot finds itself in the planned state and has only one preference over the next state given by the BT through $\mathcal{C}$. The only action which can minimize the expected free-energy is the one used during offline planning. In a nominal case, then, we maintain all the properties of BTs, which are well explained in [3]. In particular, the behavior will be finite-time successful (FTS) [3] if the atomic actions are assumed to return success after a finite time. Note that, so far we did not consider actions with the same postconditions. However, in this case, Algorithm 2 would sequentially try all the alternatives following the given order at design time. This can be improved for instance by making use of semantic knowledge at runtime to inform the action selection process about preferences over actions to achieve the same outcome. This information can be stored for instance in a knowledge base and can be used to parametrize the generative model for active inference.

2) Dynamic Environment IS NOT as Planned: The most interesting case is when a subsequent desired state is not reachable as initially planned. As explained before, in such a case we push the missing preconditions of the selected action into the current prior $\mathcal{C}$ to locally and temporarily modify the goal. We analyze this idea in terms of sequential controllers (or actions) composition as in [39], and we show how Algorithm 2 generates a plan that will eventually converge to the initial goal. First of all, we provide some assumptions and definitions that will be useful for the analysis.

Assumption 1: The action templates with pre- and postconditions provided to the agent are correct;

Assumption 2: A given desired goal is achievable by at least one atomic action.

Definition 1: The domain of attraction of an action a_i is defined as the set of its preconditions. This domain for a_i is indicated as $\mathcal{D}(a_i)$.

Definition 2: We say that an action a_1 *prepares* action a_2 if the postconditions $\mathcal{P}_c$ of a_2 lie within the domain of attraction of a_1, so $\mathcal{P}_c(a_2) \subseteq \mathcal{D}(a_1)$.

Note: For the derivations in this section we consider, without lack of generality, one single factor such that we can drop the superscripts, i.e., $\boldsymbol{C}^{(f_j)} = \boldsymbol{C}$.

Following Algorithm 2 each time a prior leaf node is ticked in the BT, active inference is used to define a sequence of actions to bring the current state toward the goal. It is sufficient to show, then, that the asymptotically stable equilibrium of a generic generated sequence is the initial given goal.

Lemma 1: Let $\mathcal{L}_c(\tau)$ be the current logic state of the world, and $\mathcal{A}$ the set of available actions. An action $a_i \in \mathcal{A}$ can only be executed within its domain of attraction, so when $\mathcal{L}_c(\tau) \in \mathcal{D}(a_i)$. Let us assume that the goal encoded in $\boldsymbol{C}$ is a postcondition of an action a_1 such that $\mathcal{P}_c(a_1) = \boldsymbol{C}$, and that $\mathcal{L}_c(\tau) \neq \boldsymbol{C}$. If $\mathcal{L}_c(\tau) \notin \mathcal{D}(a_1)$, Algorithm 2 generates a plan $\pi = \{a_1, \ldots, a_N\}$ with domain of attraction $\mathcal{D}(\pi)$ according to the steps as follows.

1) Let the initial sequence contain $a_1 \in \mathcal{A}$, $\pi(1) = \{a_1\}$, $\mathcal{D}(\pi) = \mathcal{D}(a_1)$, set $N = 1$.
2) Remove a_N from the available actions, and add the unmet preconditions $\mathcal{D}(a_N)$ to the prior $\boldsymbol{C}$ with higher priority, such that $\boldsymbol{C} = \boldsymbol{C} \cup \mathcal{D}(a_N)$.
3) Select a_{N+1} through active inference (Algorithm 1). Then, a_{N+1} prepares a_N by construction, $\pi(N+1) = \pi(N) \cup \{a_N\}$, $\mathcal{D}_{N+1}(\pi) = \mathcal{D}_N(\pi) \cup \mathcal{D}(a_{N+1})$, and $N = N + 1$.
4) Repeat 2, 3 until $\mathcal{L}_c(\tau) \in \mathcal{D}(a_N)$ `OR` a_N `==` `Idle`

If $\mathcal{L}_c(\tau) \in \mathcal{D}(a_N)$, the sequential composition π with region of attraction $\mathcal{D}(\pi) = \bigcup_{a_i} \mathcal{D}(a_i)$ for $i = 1, \ldots, N$ stabilizes the system at the given desired state $\boldsymbol{C}$. If $a_i \in \mathcal{A}$ are FTS, then π is FTS.

Proof: Since $\mathcal{L}_c(\tau) \in \mathcal{D}(a_N)$ and $\mathcal{P}_c(a_N) \subseteq \mathcal{D}(a_{N-1})$, it follows that $\mathcal{L}_c(\tau)$ is moving toward $\boldsymbol{C}$. Moreover, by construction $\mathcal{D}(a_1) \subseteq \bigcup_{a_i} \mathcal{P}_c(a_i)$ for $i = 2, \ldots, N$. After completing action a_1, it results in $\mathcal{L}_c(\tau) \equiv \boldsymbol{C}$ since by definition $\mathcal{P}_c(a_1) = \boldsymbol{C}$. ∎

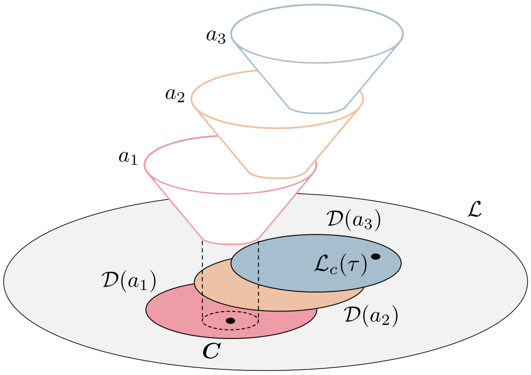


Fig. 6. Schematic visualization of the domain of attraction $\mathcal{D}(\cdot)$ of different controllers around the current logical state $\mathcal{L}_c(\tau)$, as well as their postconditions within the domain of attraction of the controller below.

Note that if $\mathcal{L}_c(\tau) \in \mathcal{D}(a_N)$ does not hold after sampling all available actions, it means that the algorithm is unable to find a set of actions that can satisfy the preconditions of the initially planned action. This situation is a major failure that needs to be handled by the overall BT. Lemma 1 is a direct consequence of the sequential behavior composition of FTS actions, where each action has effects within the domain of attraction of the action below. The asymptotically stable equilibrium of each controller is either the goal $\boldsymbol{C}$, or it is within the region of attraction of another action earlier in the sequence, see [3], [39], [40]. One can visualize the idea of sequential composition in Fig. 6.

B. Analysis of Robustness

It is not easy to find a common and objective definition of robustness to analyze the characteristics of algorithms for task execution. One possible way is to describe robustness in terms of domains or regions of attraction as in past work [3], [39]. When considering task planning and execution with classical BTs, often these regions of attraction are defined offline leading to a complex and extensive analysis of the possible contingencies that might eventually happen [3], and these are specific to each different task. Alternatively, adapting the region of attraction requires either replanning [6] or dynamic BT expansion [4]. Robustness can be measured according to the size of this region, such that a robot can achieve the desired goal from a plurality of initial conditions. With Algorithm 2 we achieve robust behavior by *dynamically* generating a suitable region of attraction according to the minimization of free-energy. This region brings the current state toward the desired goal. We then cover only the necessary region in order to be able to steer the current state to the desired goal, changing prior preferences at run-time.

Corollary 1: When an executable action a_N is found during task execution through Algorithm 2 such that $\pi = \{a_1, \ldots, a_N\}$, the plan has a domain of attraction toward a given goal that includes the current state $\mathcal{L}_c(\tau)$. If $\mathcal{D}(a_1) \subseteq \bigcup_{a_i} \mathcal{P}_c(a_i)$ for $i = 2, \ldots, N$ the plan is asymptotically stable.

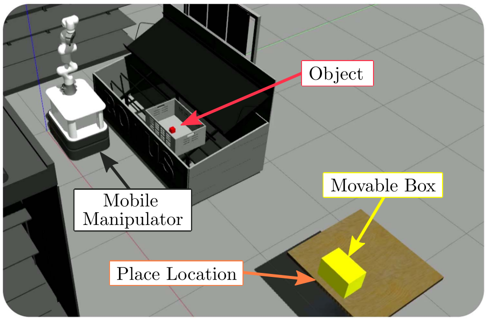


Fig. 7. Simulation of the mobile manipulation task.

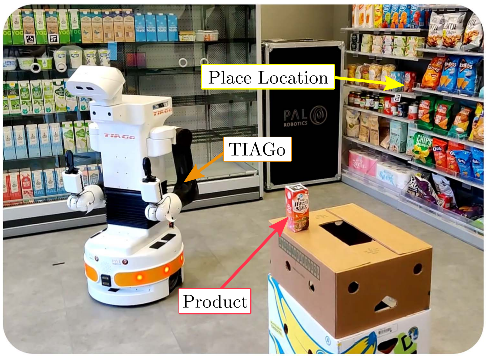


Fig. 8. Experiments with TIAGo, stocking a product on the shelf.

TABLE II
NOTATION FOR STATES AND ACTIONS

State, Boolean State	Description
$s^{(loc)}$, $l^{(loc)}$	Belief about being at the goal location
$s^{(hold)}$, $l^{(hold)}$	Belief about holding an object
$s^{(reach)}$, $l^{(reach)}$	Belief about reachability of an object
$s^{(place)}$, $l^{(place)}$	Belief about an object being placed at a location
$s^{(free)}$, $l^{(free)}$	Belief about a location being free

Actions	Preconditions	Postconditions
`moveTo(goal)`	-	$l^{(loc)} = [1\ 0]^\top$ $l^{(reach)} = [1\ 0]^\top$

`pick(obj)`	`isReachable(obj)`
	`!isHolding`
		$l^{(hold)} = [1\ 0]^\top$

`place(obj,loc)`	`isLocationFree(loc)`	$l^{(place)} = [1\ 0]^\top$
`push()`	`!isHolding`	$l^{(free)} = [1\ 0]^\top$
`placeOnPlate()`	-	$l^{(hold)} = [0\ 1]^\top$

Proof: The corollary follows simply from Lemma 1. ■

Example 9: Let us assume that Algorithm 2 produced a plan $\pi = \{a_1, a_2\}$, a set of FTS actions, where a_2 is executable so $\mathcal{L}_c(\tau) \in \mathcal{D}(a_2)$, and its effects are such that $\mathcal{P}_c(a_2) \equiv \mathcal{D}(a_1)$. Since a_2 is FTS, after a certain running time $\mathcal{L}_c(\tau) \in \mathcal{P}_c(a_2)$. The next tick after the effects of a_2 took place, $\pi = \{a_1\}$, where this time a_1 is executable since $\mathcal{L}_c(\tau) \in \mathcal{D}(a_1)$ and $\mathcal{P}_c(a_1) = \boldsymbol{C}$. The overall goal is then achieved in a finite time.

Instead of looking for globally asymptotically stable plans from each initial state to each possible goal, which can be unfeasible or at least very hard [39], we define smaller regions of attractions dynamically, according to the current state and goal.

V. Experimental Evaluation

In this section, we evaluate our algorithm in terms of robustness, safety, and conflicts resolution in two validation scenarios with two different mobile manipulators and tasks. We also provide a theoretical comparison with classical and dynamically expanded BTs.

A. Experimental Scenarios

1) Scenario 1: The task is to pick one object from a crate and place it on top of a table. This object might or might not be reachable from the initial robot configuration, and the placing location might or might not be occupied by another movable box. Crucially, the state of the table cannot be observed until the table is reached. This results in a partially observable initial state at the start of the mission where the place location has a 50% chance of being either free or occupied. In addition, we suppose that other external events, or agents, can interfere with the execution of the task, resulting in either helping or adversarial behavior. The robot used for the first validation scenario is a mobile manipulator consisting of a Clearpath Boxer mobile base, in combination with a Franka Emika Panda arm. The experiment for this scenario was conducted in a Gazebo simulation in a simplified version of a real retail store, see Fig. 7.

2) Scenario 2: The task is to fetch a product in a mockup retail store and stock it on a shelf using the real mobile manipulator TIAGo, as in Fig. 8.

Importantly, the BT for completing the task in the real store with TIAGo is the same one used for simulation with the Panda arm and the mobile base, just parametrized with a different object and place location. The code developed for the experiments and theoretical examples is publicly available.[4]

B. Implementation

1) Models for Scenarios 1 and 2: In order to program the tasks for Scenarios 1 and 2, we extended the robot skills defined in our theoretical Example 8. We added then two extra states and their relative observations: `isPlacedAt(loc, obj)` called $s^{(\text{place})}$, and `isLocationFree(loc)` called $s^{(\text{free})}$. The state $s^{(\text{place})}$ indicates whether or not `obj` is at `loc`, with associated probability, while $s^{(\text{free})}$ indicates whether `loc` is occupied by another object. Then, we also had to add three more actions, which are 1) `place(obj, loc)`, 2) `push(obj)` to free a placing location, and 3) `placeOnPlate(obj)`, to place the object held by the gripper on the robot's plate. We summarize the states and skills for the mobile manipulator in Table II.

The likelihood matrices are just the identity, while the transition matrices simply map the postconditions of actions, similarly to Example 8. Note that the design of actions and states is not unique, and other combinations are possible. One can make atomic actions increasingly more complex, or add more preconditions. The plan, specified in a BT, contains the desired

[4] [Online]. Available: https://github.com/cpezzato/discrete_active_inference

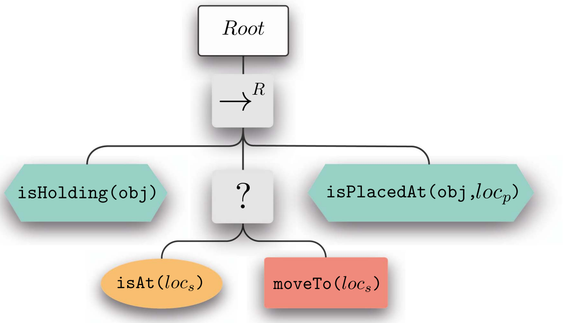
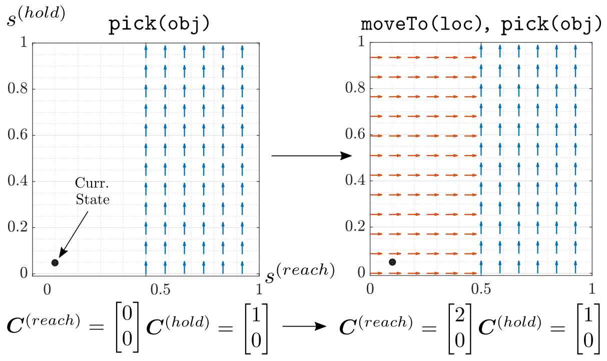


Fig. 9. BT with prior nodes to complete the mobile manipulation task in the retail store, Scenario 1 and 2. loc_s, loc_p are, respectively, the location in front of the shelf in the store and the desired place location of an item.



(a) (b)

Fig. 10. Dynamic domain of attraction generated by Algorithm 2 for Example 11. (a) relates to the action pick(obj), and (b) is the composition of moveTo(loc) and pick(obj) after automatically updating the prior preferences.

sequence of states to complete the task, leaving out from the offline planning other complex fallbacks to cope with contingencies associated with the dynamic nature of the environment. The BT for performing the tasks in both Scenario 1 and Scenario 2 is reported in Fig. 9. Note that the fallback for the action moveTo could be substituted by another prior node, as in Fig. 4; however, we opted for this alternative solution to highlight the hybrid combination of classical BTs and active inference. Design principles to choose when to use prior nodes and when normal fallbacks are reported in Section V-F.

2) Execution of Algorithm 2: We provide a full execution of Algorithm 2 in Example 10. We consider Scenario 1, for which the initial configuration of the robot is depicted in Fig. 7, and the BT for the task is the one if Fig. 9.

Example 10: Let us consider Algorithm 2 and Scenario 1. At the start of the task, the first node isHolding(obj) in the BT is ticked, and the corresponding prior preference is set, $\boldsymbol{C}^{(\text{hold})} = [1,0]^\top$ (line 2 Algorithm 2). Since the robot is not holding the desired obj and it is not reachable, $o^{(\text{hold})} = o^{(\text{reach})} = [0,1]^\top$. At the start, the initial states are a uniform distribution $s^{(hold)} = s^{(reach)} = [0.5, 0.5]^\top$ (line 4). Since the task just started, the only prior preference is the one set by the BT, so there are no high-priority priors (lines 6–10). Algorithm 1 is then run (line 12), updating the states $s^{(hold)}$, $s^{(reach)}$ according to the given observations, and selecting an action a_τ. In this example, the updated most probable logical state (line 13) will be $l_c^{(hold)} = l_c^{(reach)} = [0,1]^\top$ and a_τ will be pick(obj) since there is a mismatch between the $C^{(hold)}$ and $l_c^{(hold)}$. The preconditions of pick(obj) are checked (line 19). This action requires the object to be reachable, so this missing precondition is added to the preferences with high priority, that is $C^{(reach)} = [2,0]^\top$. Active inference is run again with the update prior (line 27). This process (lines 17–32) is repeated until either an executable action is found (lines 19–21) or the selected action is idle (lines 28–30). In the first case, a_τ is executed and the algorithm returns running. In the second case, a failure is returned indicating that no action can be performed to satisfy prior preferences. In this example, rerunning active inference (line 27) with $\boldsymbol{C}^{(hold)} = [1,0]^\top$ and $\boldsymbol{C}^{(reach)} = [2,0]^\top$ would return the action moveTo. There are no preconditions for this action, thus it can be executed (lines 20–21). The BT keeps being ticked at a certain frequency, and until the object becomes reachable, the same steps, as just described, will be repeated. As soon as the robot can reach the object, so $l_c^{(reach)} = [1,0]^\top$, the preference for $\boldsymbol{C}^{(reach)}$ is removed (lines 6–10) and set to $[0,0]^\top$. This time, when the action pick is selected, its preconditions are satisfied, and thus, it can be executed. After holding the object, when the BT is ticked no action is needed since the prior is already satisfied. Algorithm 2 returns success (lines 14–16) and the task can proceed with the next node in the BT.

Note that the BT designed for Scenario 1 was entirely reused in Scenario 2, with the only adaptation of the desired object and locations in the BT. In Section V-C and V-D, robustness and run-time conflicts resolution are analyzed for Scenario 1, but similar considerations can be derived for scenario 2.

C. Robustness: Dynamic Regions of Attraction

With our approach we improve robustness compared to classical BTs in two different ways: 1) in terms of task execution, and 2) against noisy sensory readings, which we elaborate in the following.

1) Robustness of Task Execution: With our algorithm we can generate complex regions of attraction dynamically at runtime, alleviating the burden of programming every fallback beforehand in a BT, which is prone to fail in edge cases that have not been considered at design time. We illustrate this in the following example. Consider Example 10. According to the current world's state, Algorithm 2 selects different actions to generate a suitable domain of attraction.

Example 11: The initial conditions are such that the object is not reachable. Let $s^{(hold)}$ be the probabilistic belief of holding an object, and $s^{(reach)}$ be the probabilistic belief of its reachability. The domain of attraction generated by Algorithm 2 at runtime is depicted in Fig. 10 using phase portraits, as in [3]. Actions, when performed, increase the probability of their postconditions.

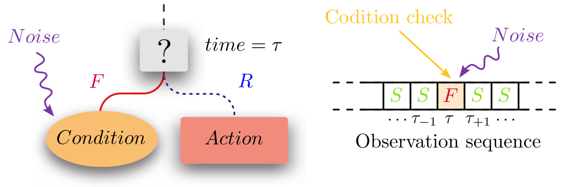


Fig. 11. Erroneous transition in a purely reactive BT due to a noisy observation at time τ. F, S, and R mean, respectively, *Failure, Success* and *Running*.

From Fig. 10, we can see that the goal of the active inference agent is to hold `obj` so $\boldsymbol{C}^{(\text{hold})} = [1\ \ 0]^\top$. The first selected action is then `pick(obj)`. However, since the current logical state is not contained in the domain of attraction of the action, the prior preferences are updated with the missing (higher priority) precondition according to the action template provided, that is `isReachable` so $\boldsymbol{C}^{(\text{reach})} = [2, 0]^\top$. This results in a sequential composition of controllers with a stable equilibrium corresponding to the postconditions of `pick(obj)`. On the other hand, to achieve the same domain of attraction with a classical BT, one would require several additional nodes, as explained in Section V-F and visualized in Fig. 14. Instead of extensively programming fallback behaviors, Algorithm 2 endows our actor with deliberation capabilities and allows the agent to reason about the current state of the environment, the desired state, and the available action templates.

2) Robustness Against Noisy Sensory Readings: BTs are purely reactive which means that every action, or subbehavior, is executed in response to an event or a condition determined at the current time step τ. If an instantaneous observation is erroneous, wrong transitions could be triggered because in classical BTs there is no notion and representation of a "state" that is maintained and updated over time. This might be a problem in the presence of noise in the observations.

Example 12: Consider a generic fallback based on a condition check in a BT, as in Fig. 11. The perception system, on which the condition check is based produced at time τ an erroneous observation, for instance, due to poor lighting conditions in an object detection algorithm. If the condition is checked at time τ, a purely reactive system would produce a wrong transition because the condition returns failure, and the fallback would tick the action.

In active inference, the probabilistic representation of a state helps filter out this kind of spurious sensory input. For instance, a wrong observation like the one above would have caused the robot to be just slightly less confident about that state, with no erroneous transition.

D. Resolving Run-Time Conflicts

Unexpected events affecting the system during action selection can lead to conflicts with the initial offline plan. This is the case in one execution of the mobile manipulation task, as in Fig. 12, where after picking the object and moving in front of the table, the robot senses that the place location is not free. In this situation, a conflict with the offline plan arises, where there is a preference for two mutually exclusive states, namely, holding the red cube but also having the gripper free in order to empty the place location. We describe this situation more formally in Example 13, and we then explain how such a conflict is resolved.

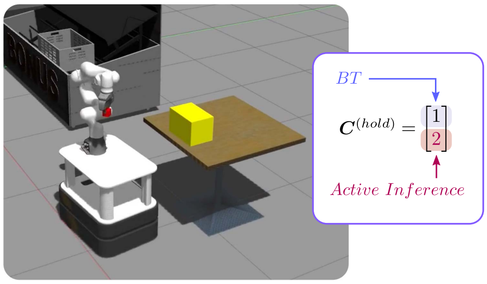

BT
$\boldsymbol{C}^{(hold)} = \begin{bmatrix}1\\2\end{bmatrix}$
Active Inference

Fig. 12. Example of conflict during mobile manipulation using the BT from Fig. 9. The BT is defining a preference of 1 over holding the red cube, but the runtime situation requires a free gripper to safely push away an unexpected object. Active inference is then required to achieve an unmet precondition with higher preference.

Example 13: For solving the situation in Fig. 12 using the BT in Fig. 9, the robot should 1) hold the object, 2) be at the desired place location, and 3) have the object placed. The preferences are planned offline and the BT populates the relative priors with a unitary preference at runtime [(14a)]. At this point of the execution, there is a mismatch between the current logical belief about the state $l^{(\text{place})}$ and the desired one, in fact $\boldsymbol{C}^{(place)}$ differs from $l^{(place)}$ [(14b)] because the object is not placed at the desired location

$$\boldsymbol{C}^{(hold)} = \begin{bmatrix}1\\0\end{bmatrix},\ \boldsymbol{C}^{(place)} = \begin{bmatrix}1\\0\end{bmatrix} \tag{14a}$$

$$l^{(hold)} = \begin{bmatrix}1\\0\end{bmatrix},\ l^{(place)} = \begin{bmatrix}0\\1\end{bmatrix}. \tag{14b}$$

The selected action with active inference in this situation is `place(obj,loc)`. The missing precondition on the place location to be free is added to the prior [see $\boldsymbol{C}^{(free)}$ in (15)] and the action selection is performed again. The only action that can minimize free-energy further is now `push`. Then, the missing precondition for this action (i.e., `!isHolding`) is added in the current prior with higher priority in $\boldsymbol{C}^{(hold)}$ (line 24 in Algorithm 2)

$$\boldsymbol{C}^{(free)} = \begin{bmatrix}2\\0\end{bmatrix},\ \boldsymbol{C}^{(hold)} = \begin{bmatrix}1\\2\end{bmatrix}. \tag{15}$$

The required `push` action to proceed with the task has a conflicting precondition with the offline plan, see $\boldsymbol{C}^{(\text{hold})}$ in (15).

Even though the desired state specified in the BT is `isHolding(obj)`, at this particular moment there is a higher preference for having the gripper free due to a missing precondition to proceed with the plan. Algorithm 2 selects then the action that best matches the current prior desires, or equivalently that minimizes expected free-energy the most, that is `placeOnPlate` to obtain $l^{(\text{hold})} = [0,\ 1]^\top$. This allows, then, to perform the action `push`. Once the place location is free after pushing, the high-priority preference on having the location free is removed from the prior. As a consequence, there are also no more preferences pushed with high priority over the state $l^{(\text{hold})}$

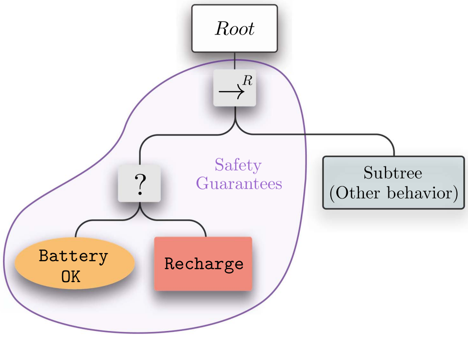


Fig. 13. BT with safety guarantees while allowing runtime adaptation.

TABLE III
SUMMARY OF COMPARISON

Approach	# Hand-crafted nodes	Unforeseen contingen.	Selective adaptation	Adaptive safety guarantees
Standard BT	27	✗	✗	✓
Dynamic BT [4]	1	✓	✗	Only for the task as a whole
Ours	6	✓	✓	✓

which is only set by the BT as $C^{(\text{hold})} = [1,\ 0]^\top$. The red cube is then picked again and placed on the table since no more conflicts are present.

Videos of the simulations and experiments can be found online.[5] The BTs to encode priors for active inference are implemented using a standard library [37].

E. Safety

When designing adaptive behaviors for autonomous robots, attention should be paid to *safety*. The proposed algorithm allows to retain control over the general behavior of the robot and to force a specific routine in case something goes wrong leveraging the structure of BTs. In fact, we are able to include adaptation through active inference only in the parts of the task that require it, keeping all the properties of BTs intact. Safety guarantees for the whole task can easily be added by using a sequence node, where the leftmost part is the safety criteria to be satisfied [3] by the right part of the behavior, as shown in Fig. 13. In this example, the BT allows avoiding battery drops below a safety-critical value while performing a task. The subtree on the right can be any other BT, for instance, the one used to solve Scenario 1 and Scenario 2 from Fig. 9.

Since, by construction, a BT is executed from left to right, one can assure that the robot is guaranteed to satisfy the leftmost condition first before proceeding with the rest of the behavior. In our specific case, this allows us to easily override the online decision-making process with active inference where needed, in favor of safety routines. Note that safety guarantees can also be provided in specific parts of the three only, and not necessarily for the whole tree. For example in Fig. 9, one might ensure navigation at a low speed only while transporting an object to a place location.

F. Comparison and Design Principles

1) Comparison With Other BT Approaches: The hybrid scheme of active inference and BTs aims at providing a framework for reactive action planning and execution in robotic systems. For this reason, we compare the properties of our approach with standard BTs [3] and with BTs generated through expansion from goal conditions [4]. Scenario 1 and Scenario 2 can be tackled, for instance, by explicitly planning every fallback behavior with classical BTs, as in Fig. 14. Even if this provides the same reactive behavior as the one generated by Fig. 9, far more (planning) effort is needed: to solve the same task one would require 12 control nodes, 8 condition nodes, and 7 actions, for a total of 27 nodes compared to the 6 needed in our approach that is an ∼88% compression.

Importantly, the development effort of a prior node in a BT is the same as a standard action node. It is true that active inference requires specifying the likelihood and transition matrices encoding actions pre- and postconditions, but this has to be done only once while defining the available skills of a robot, and it is independent of the task to be solved. Thus, a designer is not concerned with this when adding a prior node in a BT.

Instead of planning several fallbacks offline, Colledanchise et al. [4] dynamically expanded a BT from a single goal condition, through backchaining, to blend planning and acting online. To solve Scenario 1 and Scenario 2 with this approach, one needs to define a goal condition `isPlacedAt(obj, loc)` similarly to our solution, and define the preconditions of the action `place(obj, loc)` such that they contain the fact that the robot is holding the object, that the place location is reachable, and it is free. Then, to solve Scenarios 1 and 2 one needs to define only the final goal condition and run the algorithm proposed in [4]. Even though this allows to complete tasks similar to what we propose,Colledanchise et al. [4] comes with a fundamental theoretical limitation: adaptation cannot be selectively added only to specific parts of the tree. The whole behavior is indeed determined at runtime based on preconditions and effects of actions starting from a goal condition. The addition of safety guarantees can only happen for the whole task, and not in selected parts of the tree derived online.

To conclude, the hybrid combination of active inference and BTs allows for combining the advantages of both offline design of BTs and online dynamic expansion. In particular, it drastically reduces the number of necessary nodes planned offline in a BT, it can handle partial observability of the initial state, and it allows to selectively add adaptation and safety guarantees in specific parts of the tree.

Another important difference between our approach and other BTs solutions is that we introduced the concept of *state* in a BT through the *prior* node for which a probabilistic belief is built, updated, and used for action planning at runtime with uncertain action outcomes.

[5] [Online]. Available: https://youtu.be/dEjXu-sD1SI

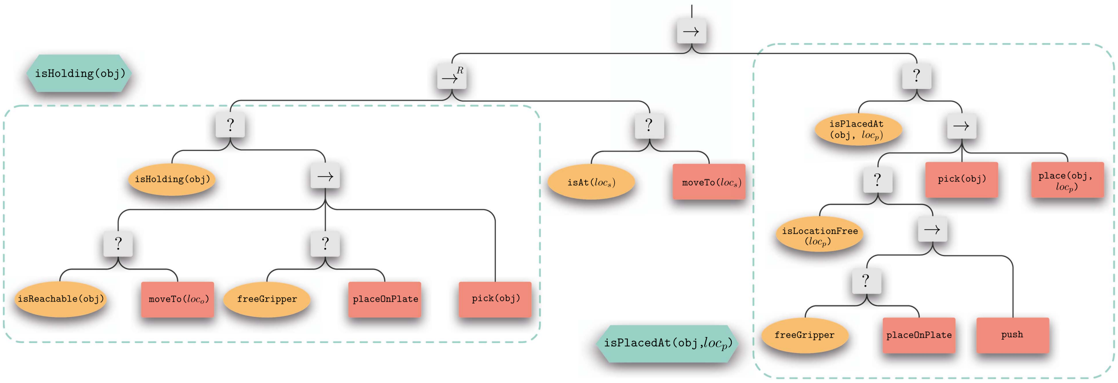


Fig. 14. Possible standard BT to perform Scenario 1 and Scenario 2 without prior nodes for active inference. Parts of the behavior that require several fallbacks can be substituted by prior nodes for online adaptation instead.

Table III reports a summary of the comparison with standard BTs and BT with dynamic expansion for Scenarios 1 and 2.

2) Comparison With ROSPlan: One may argue that other solutions, such as ROSPlan [41] could also be used for planning and execution in robotics in dynamic environments. ROSPlan leverages automated planning with PDDL2.1, but it is not designed for fast reaction in case of dynamic changes in the environment caused by external events, and would not work in case of a partially observable initial state. Consider Scenario 1. At the start of the task, the robot can sense that the gripper is empty and it has access to its current base location. However, it has no information about the state of the table (occupied or free) on which the red cube needs to be placed. At the start of the mission, the table has then 50% chance of being occupied and a 50% chance of being free, since it cannot be observed. Yet, the whole task can be planned at a high level, as in Fig. 9, and be executed. Once the robot reaches the placing location, observations regarding the state of the table become available and the internal beliefs can be updated until enough evidence is collected and a decision can be taken. Without knowing the full state at the start, a solution with ROSPlan would require making assumptions on the value of the unknown states, and either planning for the worst-case scenario, which might not even be needed, or failing during execution and replan.

3) Design Principles: We position our work in between two extremes, namely, fully offline planning and fully online dynamic expansion of BTs. In our method, a designer can decide if to lean toward a fully offline approach or a fully online synthesis. The choice depends on the task at hand and the modeling of the actions pre- and postconditions. Even though the design of behaviors is still an art, we give some design principles which can be useful in the development of robotic applications using this hybrid BTs and active inference method. Take for instance Figs. 9 and 14. Prior nodes for local adaptation can be included in the behavior when there are several contingencies to consider or action preconditions to be satisfied in order to achieve a subgoal. A designer can: 1) plan offline where the task is certain or equivalently where a small number of things can go wrong; 2) use prior nodes implemented with active inference to decide at runtime the actions to be executed whenever the task is uncertain. This is a compromise between a fully defined plan where the behavior of the robot is predefined in every part of the state space and a fully dynamic expansion of BTs which can result in a suboptimal action sequence [4]. This is illustrated in Fig. 9, where the actions for holding and placing an object are chosen online due to various possible unexpected contingencies, whereas the `moveTo` action is planned. Prior nodes should be used whenever capturing the variability of a part of a certain task would require much effort during offline planning.

VI. Discussion

In this work, we considered a mobile manipulator in a retail store domain with particular focus on plan execution. In this scope, offline planning is arguably better suited to offload computations at runtime for the parts of the task that do not change frequently. In our case, this was the sequence of states to stock a product that was encoded in a BT. On the other hand, at the cost of additional online computations, local online planning is better suited for execution in uncertain environments, such as a busy supermarket, because one can avoid planning beforehand for every contingency. We achieved this through active inference. With the proposed method we can leverage the complementary advantages of both offline and online planning, and in the following we discuss in detail the choice of using BTs and active inference specifically.

A. Active Inference as a Planning Node in a BT

We opted for the use of active inference with action preconditions as a planning node in the BT because this allows achieving online PDDL-style planning with noisy observations and partially observable probabilistic states. An alternative solution to our approach could be to use a simple PDDL planner in conjunction with a filtering scheme. By replanning for a small subtask at the same frequency used for the active inference node, one can achieve similar reactiveness to our approach. However, by means of active inference, one can make use of the full probabilistic information on the states, and one does not require full knowledge of the symbolic initial state.

B. Why Choosing BTs

An experienced roboticist could also wonder why we opted for BTs in the first place, instead of other PDDL-style planning approaches to encode the solution to a task. First of all, the focus of this article is on the runtime adaptability and reactivity in dynamic environments with partially observable initial state, and not on the generation of complex offline plans.

In this context, BTs are advantageous because they are designed to dispatch and monitor actions execution at runtime. This allows to quickly react to changes in the environment that are not necessarily a consequence of the robot's own actions. A plan that is generated through PDDL planning and executed bypassing BTs cannot provide this type of reactivity unless one defines specific replanning strategies to mimic BTs' reactiveness.

However, we see potential extensions of this work by combining it with other PDDL planning methods. For instance, PDDL can be used to automatically generate plans for which their execution is optimized through BTs [42]. This allows for instance to take advantage of parallel action execution and would remove the need to hand-design a BT. In additional, an action at runtime can be executed as soon as its requirements are available instead of waiting for what is established offline by the planner.

C. Why Choose Active Inference

Active inference could potentially be substituted by other valid POMDP approaches. We see two possible ways of doing so, that is either using an *offline* or an *online* POMDP solver.

First, one could solve a policy *offline* and then use it for online decision-making. This approach can be more effective than active inference once the transition matrices, the reward, and the task are fixed. However, the addition of new symbolic actions (so new skills as transitions), or a substantial change in the task while using the same skills, would require recomputing the policy. In addition, planning offline for all possible states and actions combinations is a much larger problem than computing a plan online for the current state only. As concluded in [43], for the parts of a task subject to frequent unpredictable changes, computing locally an online plan as we do with active inference is preferable to offline policies.

Second, one could perform *online* decision-making without offline computations and achieve similar performance, see [44] and [45]. Compared to both offline and online POMDPs, however, active inference exposes extra model parameters to bridge abstract common sense knowledge with discrete decision-making. These extra model parameters can be updated with runtime information to adapt plans on the fly. Take for instance the prior over plans $p(\pi) = \mathrm{Cat}(\boldsymbol{E})$. The $\boldsymbol{E}$ vector can be updated at runtime to steer the decision-making while computing the posterior distribution $\boldsymbol{\pi} = \sigma(\ln \boldsymbol{E} - \boldsymbol{G}_\pi - \boldsymbol{F}_\pi)$ [34]. This vector can be used to encode common sense and habits [34], [35], but can also be used to adapt the plans online due for instance to runtime component failure. This opens up many possibilities for extensions, to be explored in future work. We are particularly interested in active inference because it is a flexible and unified framework that connects different branches of control theory at different abstraction levels. Active inference can unify 1) abstract decision-making with guarantees, as in this article, 2) adaptive [12] and fault tolerant [10], [11], [46] torque control, as well as 3) state estimation and learning.

VII. Conclusion

In this work, we tackled the problem of action planning and execution in real-world robotics. We have addressed two open challenges, namely, hierarchical deliberation, and continual online planning, by combining BTs and active inference. The proposed algorithm and its core idea are general and independent of the particular robot platform and task. Our solution provides local reactivity to unforeseen situations while keeping the initial plan intact. In addition, it is possible to easily add safety guarantees to override the online decision-making process thanks to the properties of BTs. We showed how robotic tasks can be described in terms of free-energy minimization, and we introduced action preconditions and conflict resolution for active inference by means of dynamic priors. This means that a robot can locally set its own sub-goals to resolve a local inconsistency, and then return to the initial plan specified in the BT. We performed a theoretical analysis of the convergence and robustness of the algorithm, and the effectiveness of the approach is demonstrated on two different mobile manipulators and different tasks, both in simulation and real experiments.

Appendix A
Generative Models

Consider the generative model in active inference $P(\bar{o}, \bar{s}, \boldsymbol{\eta}, \pi)$. By using the chain rule, we can write

$$P(\bar{o}, \bar{s}, \boldsymbol{\eta}, \pi) = P(\bar{o}|\bar{s}, \boldsymbol{\eta}, \pi)P(\bar{s}|\boldsymbol{\eta}, \pi)P(\boldsymbol{\eta}|\pi)P(\pi). \quad (16)$$

Note that $\bar{o}$ is conditionally independent from the model parameters $\boldsymbol{\eta}$ and π given $\bar{s}$. In addition, under the Markov property, the next state and current observations depend only on the current state

$$P(\bar{o}|\bar{s}, \boldsymbol{\eta}, \pi) = \prod_{\tau=1}^{T} P(o_\tau|s_\tau). \quad (17)$$

The model is further simplified considering that $\bar{s}$ and $\boldsymbol{\eta}$ are conditionally independent given π

$$P(\bar{s}|\boldsymbol{\eta}, \pi) = \prod_{\tau=1}^{T} P(s_\tau|s_{\tau-1}, \pi). \quad (18)$$

Finally, consider the model parameters explicitly

$$\begin{aligned} P(\bar{o}, \bar{s}, \boldsymbol{\eta}, \pi) &= P(\bar{o}, \bar{s}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{D}, \pi) \\ &= P(\pi)P(\boldsymbol{A})P(\boldsymbol{B})P(\boldsymbol{D}) \prod_{\tau=1}^{T} P(s_\tau|s_{\tau-1}, \pi)P(o_\tau|s_\tau). \end{aligned} \quad (19)$$

$P(\boldsymbol{A}), P(\boldsymbol{B}), P(\boldsymbol{D})$ are the Dirichlet distributions over the model parameters, [20]. In case the model parameters are fixed by the user, as in this work, it holds

$$P(\bar{o}, \bar{s}, \pi) = P(\pi) \prod_{\tau=1}^{T} P(s_\tau | s_{\tau-1}, \pi) P(o_\tau | s_\tau). \tag{20}$$

Given the generative model above, we are interested in finding the posterior hidden causes of sensory data. For the sake of these derivations, we consider that the parameters associated with the task are known and do not introduce uncertainty. Using the Bayes rule

$$P(\bar{s}, \pi | \bar{o}) = \frac{P(\bar{o}|\bar{s}, \pi) P(\bar{s}, \pi)}{P(\bar{o})}. \tag{21}$$

Computing the model evidence $P(\bar{o})$ exactly is a well known and often an intractable problem in the Bayesian statistics. The exact posterior is then computed minimizing the Kullback–Leibler divergence (D_{KL}, or KL-divergence) with respect to an approximate posterior distribution $Q(\bar{s}, \pi)$. Doing so, we can define the free-energy as a functional of approximate posterior beliefs which results in an upper bound on surprise. By definition D_{KL} is a nonnegative quantity given by the expectation of the logarithmic difference between $Q(\bar{s}, \pi)$ and $P(\bar{s}, \pi|\bar{o})$. Applying the KL-divergence

$$\begin{aligned} &D_{KL}\left[Q(\bar{s}, \pi) || P(\bar{s}, \pi|\bar{o})\right] \\ &= \mathbb{E}_{Q(\bar{s},\pi)}\left[\ln Q(\bar{s}, \pi) - \ln P(\bar{s}, \pi|\bar{o})\right] \geq 0 \end{aligned} \tag{22}$$

D_{KL} is the information loss when Q is used instead of P. Considering (21) and the chain rule, (22) can be rewritten as

$$\begin{aligned} D_{KL}\left[\cdot\right] &= \mathbb{E}_{Q(\bar{s},\pi)}\left[\ln Q(\bar{s}, \pi) - \ln \frac{P(\bar{o}, \bar{s}, \pi)}{P(\bar{o})}\right] \\ &= \underbrace{\mathbb{E}_{Q(\bar{s},\pi)}\left[\ln Q(\bar{s}, \pi) - \ln P(\bar{o}, \bar{s}, \pi)\right]}_{F[Q(\bar{s},\pi)]} + \ln P(\bar{o}). \end{aligned} \tag{23}$$

We have just defined the free-energy as the upper bound of surprise

$$F\left[Q(\bar{s}, \pi)\right] \geq -\ln P(\bar{o}). \tag{24}$$

Appendix B

Variational Free-Energy

To fully characterize the free-energy in (23), we need to specify a form for the approximate posterior $Q(\bar{s}, \pi)$. There are different ways to choose a family of probability distributions [47], compromising between complexity and accuracy of the approximation. In this work, we choose the mean-field approximation. It holds

$$Q(\bar{s}, \pi) = Q(\bar{s}|\pi) Q(\pi) = Q(\pi) \prod_{\tau=1}^{T} Q(s_\tau|\pi). \tag{25}$$

Under mean-field approximation, the plan-dependent states at each time step are approximately independent of the states at any other time step. We can now find an expression for the variational free-energy. Considering the mean-field approximation and the generative model in (20) we can write

$$\begin{aligned} F\left[Q(\bar{s}, \pi)\right] = \mathbb{E}_{Q(\bar{s},\pi)}\Bigg[&\ln Q(\pi) + \sum_{\tau=1}^{T} \ln Q(s_\tau|\pi) \\ &- \ln P(\pi) - \sum_{\tau=1}^{T} \ln P(s_\tau|s_{\tau-1}, \pi) - \sum_{\tau=1}^{T} \ln P(o_\tau|s_\tau)\Bigg]. \end{aligned} \tag{26}$$

Since $Q(\bar{s}, \pi) = Q(\bar{s}|\pi)Q(\pi)$, and since the expectation of a sum is the sum of the expectation, we can write

$$F\left[\cdot\right] = D_{KL}\left[Q(\pi)||P(\pi)\right] + \mathbb{E}_{Q(\pi)}\left[F(\pi)\left[Q(\bar{s}|\pi)\right]\right] \tag{27}$$

where

$$\begin{aligned} F(\pi)\left[Q(\bar{s}|\pi)\right] = \mathbb{E}_{Q(\bar{s}|\pi)}\Bigg[&\sum_{\tau=1}^{T} \ln Q(s_\tau|\pi) \\ &- \sum_{\tau=1}^{T} \ln P(s_\tau|s_{t-\tau}, \pi) - \sum_{\tau=1}^{T} \ln P(o_\tau|s_\tau)\Bigg]. \end{aligned} \tag{28}$$

One can notice that $F(\pi)$ is accumulated over time, or in other words, it is the sum of free energies over time and plans

$$F(\pi) = \sum_{\tau=1}^{T} F(\pi, \tau). \tag{29}$$

Substituting the agent's belief about the current state at time τ given π with $\boldsymbol{s}_\tau^\pi$, we obtain a matrix form for $F(\pi, \tau)$ that we can compute given the generative model

$$F(\pi) = \sum_{\tau=1}^{T} \boldsymbol{s}_\tau^{\pi\top}\left[\ln \boldsymbol{s}_\tau^\pi - \ln\left(\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}_{\tau-1}^\pi\right) - \ln\left(\boldsymbol{A}^\top \boldsymbol{o}_\tau\right)\right]. \tag{30}$$

Given a plan π, the probability of state transition $P(s_\tau|s_{\tau-1}, \pi)$ is given by the transition matrix under plan π at time τ, multiplied by the probability of the state at the previous time step. In the special case of $\tau = 1$, we can write

$$F(\pi, 1) = \boldsymbol{s}_1^{\pi\top}\left[\ln \boldsymbol{s}_1^\pi - \ln \boldsymbol{D} - \ln\left(\boldsymbol{A}^\top \boldsymbol{o}_1\right)\right]. \tag{31}$$

Finally, we can compute the expectation of the plan-dependant variational free-energy $F(\pi)$ as $\mathbb{E}_{Q(\pi)}[F(\pi)] = \boldsymbol{\pi}^\top \boldsymbol{F}_\pi$. We indicate $\boldsymbol{F}_\pi = (F(\pi_1), F(\pi_2)\ldots)^\top$ for every allowable plan. To derive state and plan updates that minimize free-energy, F in (27) is partially differentiated and set to zero, as we will see in the following appendixes.

Appendix C

State Estimation

We differentiate F with respect to the sufficient statistics of the probability distribution of the states. Note that the only part of F dependent on the states is $F(\pi)$. Then

$$\begin{aligned} \frac{\partial F}{\partial \boldsymbol{s}_\tau^\pi} = \frac{\partial F}{\partial F(\pi)} \frac{\partial F(\pi)}{\partial \boldsymbol{s}_\tau^\pi} = \boldsymbol{\pi}^\top \big[&\mathbf{1} + \ln \boldsymbol{s}_\tau^\pi - \ln\left(\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}_{\tau-1}^\pi\right) \\ &- \ln\left(\boldsymbol{B}_{a_\tau}^\top \boldsymbol{s}_{\tau+1}^\pi\right) - \ln\left(\boldsymbol{A}^\top \boldsymbol{o}_\tau\right)\big]. \end{aligned} \tag{32}$$

Setting the gradient to zero and using the softmax function for normalization

$$\boldsymbol{s}_\tau^\pi = \sigma\left(\ln\left(\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}_{\tau-1}^\pi\right) + \ln\left(\boldsymbol{B}_{a_\tau}^\top \boldsymbol{s}_{\tau+1}^\pi\right) + \ln\left(\boldsymbol{A}^\top \boldsymbol{o}_\tau\right)\right). \tag{33}$$

Note that the softmax function is insensitive to the constant $\mathbf{1}$. Also, for $\tau = 1$ the term $\ln\left(\boldsymbol{B}_{a_{\tau-1}} \boldsymbol{s}_{\tau-1}^\pi\right)$ is replaced by $\boldsymbol{D}$. Finally, $\ln\left(\boldsymbol{A}^\top \boldsymbol{o}_\tau\right)$ contributes only to past and present time steps, so for this term is null for $t < \tau \leq T$ since those observations are still to be received.

Appendix D
Expected Free-Energy

We indicate with $G(\pi)$ the expected free-energy obtained over future time steps until the time horizon T while following a plan π. Basically, this is the variational free-energy of future trajectories which measures the plausibility of plans according to future predicted observations [21]. To compute it we take the expectation of variational free-energy under the posterior predictive distribution $P(o_\tau|s_\tau)$. Following [21] we can write

$$G(\pi) = \sum_{\tau=t+1}^{T} G(\pi, \tau) \tag{34}$$

then

$$\begin{aligned} G(\pi,\tau) &= \mathbb{E}_{\tilde{Q}}\left[\ln Q(s_\tau|\pi) - \ln P(o_\tau, s_\tau|s_{\tau-1})\right] \\ &= \mathbb{E}_{\tilde{Q}}\left[\ln Q(s_\tau|\pi) - \ln P(s_\tau|o_\tau, s_{\tau-1}) - \ln P(o_\tau)\right] \end{aligned} \tag{35}$$

where $\tilde{Q} = P(o_\tau|s_\tau)Q(s_\tau|\pi)$. The expected free-energy is

$$G(\pi,\tau) \geq \mathbb{E}_{\tilde{Q}}\left[\ln Q(s_\tau|\pi) - \ln Q(s_\tau|o_\tau, s_{\tau-1}, \pi) - \ln P(o_\tau)\right]. \tag{36}$$

Equivalently, we can express the expected free-energy in terms of preferred observations [33]

$$G(\pi,\tau) = \mathbb{E}_{\tilde{Q}}\left[\ln Q(o_\tau|\pi) - \ln Q(o_\tau|s_\tau, s_{\tau-1}, \pi) - \ln P(o_\tau)\right]. \tag{37}$$

Making use of $Q(o_\tau|s_\tau,\pi) = P(o_\tau|s_\tau)$ since the predicted observations in the future are only based on $\boldsymbol{A}$ which is plan independent given s_τ, we have

$$G(\pi,\tau) = \underbrace{D_{KL}\left[Q(o_\tau|\pi)||P(o_\tau)\right]}_{\text{Expected cost}} + \underbrace{\mathbb{E}_{Q(s_\tau|\pi)}\left[H(P(o_\tau|s_\tau))\right]}_{\text{Entropy}} \tag{38}$$

where $H[P(o_\tau|s_\tau)] = \mathbb{E}_{P(o_\tau|s_\tau)}[-\ln P(o_\tau|s_\tau)]$ is the entropy. We are now ready to express the expected free-energy in matrix form, such that we can compute it. From the previous equation, one can notice that plan selection aims at minimizing the expected cost and ambiguity. The latter relates to the uncertainty about future observations given hidden states. In a sense, plans tend to bring the agent to future states that generate unambiguous information over states. On the other hand, the cost is the difference between predicted and prior beliefs about final states. Plans are more likely if they minimize cost, and lead to observations that match prior desires. Minimizing G leads to both exploitative (cost minimizing) and explorative (ambiguity minimizing) behavior. This results in a balance between goal-oriented and novelty-seeking behaviors

Substituting the sufficient statistics in (38), and recalling that the generative model specifies $P(o_\tau) = \boldsymbol{C}$, one obtains [34]

$$G(\pi,\tau) = \underbrace{\boldsymbol{o}_\tau^{\pi\top}\left[\ln \boldsymbol{o}_\tau^\pi - \ln \boldsymbol{C}\right]}_{\text{Reward seeking}} \underbrace{-\mathrm{diag}(\boldsymbol{A}^\top \ln \boldsymbol{A})^\top \boldsymbol{s}_\tau^\pi}_{\text{Information seeking}}. \tag{39}$$

Note that prior preferences are passed through the softmax function before computing the logarithm.

Appendix E
Updating Plan Distribution

The update rule for the distribution over possible plans follows directly from the variational free-energy

$$F\left[\cdot\right] = D_{KL}\left[Q(\pi)||P(\pi)\right] + \boldsymbol{\pi}^\top \boldsymbol{F}_\pi. \tag{40}$$

The first term of the equation above can be written as

$$D_{KL}\left[Q(\pi)||P(\pi)\right] = \mathbb{E}_{Q(\pi)}\left[\ln Q(\pi) - \ln P(\pi)\right]. \tag{41}$$

Recalling that the approximate posterior over policies is a softmax function of the expected free-energy $Q(\pi) = \sigma(-G(\pi))$ [20], [33], and taking the gradient with respect to $\boldsymbol{\pi}$ it results

$$\frac{\partial F}{\partial \boldsymbol{\pi}} = \ln \boldsymbol{\pi} + \boldsymbol{G}_\pi + \boldsymbol{F}_\pi + \boldsymbol{1} \tag{42}$$

where $\boldsymbol{G}_\pi = (G(\pi_1), G(\pi_2), \ldots)^\top$. Finally, setting the gradient to zero and normalizing through softmax, the posterior distribution over plans is obtained

$$\boldsymbol{\pi} = \sigma(-\boldsymbol{G}_\pi - \boldsymbol{F}_\pi). \tag{43}$$

The plan that an agent should pursue is the most likely one.

References

[1] M. Ghallab, D. Nau, and P. Traverso, "The actor's view of automated planning and acting: A position paper," *Artif. Intell.*, vol. 208, pp. 1–17, 2014.

[2] D. S. Nau, M. Ghallab, and P. Traverso, "Blended planning and acting: Preliminary approach, research challenges," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 4047–4051.

[3] M. Colledanchise and P. Ogren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 372–389, Apr. 2017.

[4] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior tree," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 8839–8845.

[5] E. Safronov, M. Colledanchise, and L. Natale, "Task planning with belief behavior trees," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 6870–6877.

[6] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 5588–5595.

[7] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 5678–5684.

[8] A. Meera and M. Wisse, "Free energy principle based state and input observer design for linear systems with colored noise," in *Proc. Amer. Control Conf.*, 2020, pp. 5052–5058.

[9] M. Baioumy, P. Duckworth, B. Lacerda, and N. Hawes, "Active inference for integrated state-estimation, control, and learning," in *Proc. Int. Conf. Robot. Automat.*, 2021, pp. 4665–4671.

[10] C. Pezzato, M. Baioumy, C. H. Corbato, N. Hawes, M. Wisse, and R. Ferrari, "Active inference for fault tolerant control of robot manipulators with sensory faults," in *Proc. Int. Workshop Act. Inference*, 2020, pp. 20–27.

[11] M. Baioumy, C. Pezzato, R. Ferrari, C. H. Corbato, and N. Hawes, "Fault-tolerant control of robot manipulators with sensory faults using unbiased active inference," in *Proc. Eur. Control Conf.*, 2021, pp. 1119–1125.

[12] C. Pezzato, R. Ferrari, and C. H. Corbato, "A novel adaptive controller for robot manipulators based on active inference," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 2973–2980, Apr. 2020.

[13] G. Oliver, P. Lanillos, and G. Cheng, "An empirical study of active inference on a humanoid robot," *IEEE Trans. Cogn. Devel. Syst.*, vol. 14, no. 2, pp. 462–471, Jun. 2022.

[14] K. J. Friston, "The free-energy principle: A unified brain theory?," *Nature Rev. Neurosci.*, vol. 11, no. 2, pp. 27–138, 2010.

[15] C. Buckley, C. Kim, S. McGregor, and A. Seth, "The free energy principle for action and perception: A mathematical review," *J. Math. Psychol.*, vol. 81, pp. 55–79, 2017.

[16] R. Bogacz, "A tutorial on the free-energy framework for modelling perception and learning," *J. Math. Psychol.*, vol. 76, pp. 198–211, 2017.

[17] K. J. Friston, J. Mattout, and J. Kilner, "Action understanding and active inference," *Biol. Cybern.*, vol. 104, no. 1/2, pp. 137–160, 2011.

[18] K. J. Friston, J. Daunizeau, and S. Kiebel, "Action and behavior: A free-energy formulation," *Biol. Cybern.*, vol. 102, pp. 227–260, 2010.

[19] K. Friston, S. Samothrakis, and R. Montague, "Active inference and agency: Optimal control without cost functions," *Biol. Cybern.*, vol. 106, no. 8/9, pp. 523–541, 2012.

[20] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, and G. Pezzulo, "Active inference: A process theory," *Neural Computation*, vol. 29, no. 1, pp. 1–49, 2017.

[21] N. Sajid, P. J. Ball, T. Parr, and K. J. Friston, "Active inference: Demystified and compared," *Neural Computation*, vol. 33, no. 3, pp. 674–712, 2021.

[22] P. Schwartenbeck, J. Passecker, T. U. Hauser, T. H. FitzGerald, M. Kronbichler, and K. J. Friston, "Computational mechanisms of curiosity and goal-directed exploration," *Elife*, vol. 8, 2019, Art. no. e41703.

[23] R. Kaplan and K. J. Friston, "Planning and navigation as active inference," *Biol. Cybern.*, vol. 112, no. 4, pp. 323–343, 2018.

[24] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton, FL, USA: CRC Press, 2018.

[25] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 2718–2725.

[26] J. Orkin, "Applying goal-oriented action planning to games," *AI Game Program. Wisdom*, vol. 2, pp. 217–228, 2003.

[27] J. Orkin, "Three states and a plan: The AI of FEAR," in *Proc. Game Developers Conf.*, 2006, pp. 1–18.

[28] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 1470–1477.

[29] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. J. Robot. Res.*, vol. 32, pp. 1194–1227, 2013.

[30] M. Levihn, L. P. Kaelbling, T. Lozano-Pérez, and M. Stilman, "Foresight and reconsideration in hierarchical planning and execution," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2013, pp. 224–231.

[31] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *Proc. 12th Nat. Conf. Artif. Intell.*, 1994, pp. 1123–1128.

[32] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[33] L. Da Costa, T. Parr, N. Sajid, S. Veselic, V. Neacsu, and K. Friston, "Active inference on discrete state-spaces: A synthesis," *J. Math. Psychol.*, vol. 99, 2020, Art. no. 102447.

[34] R. Smith, K. J. Friston, and C. J. Whyte, "A step-by-step tutorial on active inference and its application to empirical data," *J. Math. Psychol.*, vol. 107, 2022, Art. no. 102632. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022249621000973

[35] C. Hesp, R. Smith, T. Parr, M. Allen, K. J. Friston, and M. J. Ramstead, "Deeply felt affect: The emergence of valence in deep active inference," *Neural Computation*, vol. 33, no. 2, pp. 398–446, 2021.

[36] C. Heins et al., "pymdp: A Python library for active inference in discrete state spaces," *J. Open Source Softw.*, vol. 7, no. 73, 2022, Art. no. 4098.

[37] D. Faconti, "BehaviorTree.CPP." [Online]. Available: https://www.behaviortree.dev/

[38] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, J. O'Doherty, and G. Pezzulo, "Active inference and learning," *Neurosci. Biobehavioral Rev.*, vol. 68, pp. 862–879, 2016.

[39] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *Int. J. Robot. Res.*, vol. 18, no. 6, pp. 534–555, 1999.

[40] E. Najafi, R. Babuška, and G. A. Lopes, "An application of sequential composition control to cooperative systems," in *Proc. 10th Int. Workshop Robot Motion Control*, 2015, pp. 15–20.

[41] M. Cashmore et al., "ROSPlan: Planning in the robot operating system," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2015, pp. 333–341.

[42] F. Martín, M. Morelli, H. Espinoza, F. J. Lera, and V. Matellán, "Optimized execution of PDDL plans using behavior trees," in *Proc. 20th Int. Conf. Auton. Agents MultiAgent Syst.*, 2021, pp. 1596–1598.

[43] M. Baioumy, B. Lacerda, P. Duckworth, and N. Hawes, "On solving a stochastic shortest-path Markov decision process as probabilistic inference," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2021, pp. 819–829.

[44] S. Paquet, L. Tobin, and B. Chaib-Draa, "An online POMDP algorithm for complex multiagent environments," in *Proc. 4th Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2005, pp. 970–977.

[45] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," *J. Artif. Intell. Res.*, vol. 58, pp. 231–266, 2017.

[46] M. Baioumy, C. Pezzato, C. H. Corbato, N. Hawes, and R. Ferrari, "Towards stochastic fault-tolerant control using precision learning and active inference," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2021, pp. 681–691.

[47] S. Schwöbel, S. Kiebel, and D. Marković, "Active inference, belief propagation, and the Bethe approximation," *Neural Computation*, vol. 30, no. 9, pp. 2530–2567, 2018.

Corrado Pezzato received the B.Sc. degree (with hons.) in automation engineering with the Alma Mater Studiorum, Bologna, Italy, in 2017, the M.Sc. degree in systems and control (with hons.), in 2019, with the Delft University of Technology, Delft, Netherlands, where he is currently working toward the Ph.D. degree in robotics.

He is currently with AIRLab, the AI for Retail Lab in Delft. His research interests include low-level control, high-level decision-making, and their interconnection, with a strong focus on robotics and active inference.

Carlos Hernández Corbato received the graduate degree (with hons.) in industrial engineering and the M.Sc. and Ph.D. degrees in automation and robotics from the Universidad Politécnica de Madrid, Madrid, Spain, in 2006, 2008, and 2013, respectively.

He is currently an Assistant Professor of Cognitive Robotics with the Delft University of Technology, Delft, the Netherlands. He has coordinated and participated in European projects on cognitive robotics and factories of the future. His research interests include self-adaptive systems, knowledge representation and reasoning, model-based system engineering.

Stefan Bonhof is born in Rotterdam, The Netherlands in 1996. He received the B.Sc. degree in mechanical engineering and the M.Sc. degree (with hons.) in vehicle engineering (specializing in autonomy) from the Delft University of Technology, Delft, the Netherlands, in 2018 and 2020, respectively.

He is currently the Team Lead for students following the M.Sc. Robotics program doing their thesis at AIRLab Delft, as well as technical support and robotics engineer for the whole lab.

Martijn Wisse received the M.Sc. and Ph.D. degrees in mechanical engineering from the Delft University of Technology, Delft, The Netherlands, in 2000 and 2004, respectively.

He is currently a Professor with the Delft University of Technology. His previous research focused on passive dynamic walking robots and passive stability in the field of robot manipulators. He worked on underactuated grasping, open-loop stable manipulator control, design of robotic systems, and the creation of startups in this field. His research interests focus on the neuroscientific principle of active inference and its application and advancements in robotics.