Towards a symbolic implementation of Active Inference for Lego robots

Jean-François Cloutier, SmartRent
Last updated: July 19, 2022

Abstract

I program autonomous Lego robots to explore concepts of cognition and make these concepts tangible. My goal is for robots to conform to the principles of Active Inference and learn as much as possible on their own. As a software developer and not a neuroscientist, I take an unorthodox approach. Instead of encoding the mathematics of the Free Energy Principle, I implement models of cognition as collections of concurrent processes that carry out "symbolic" computations and communicate via events, something akin to Marvin Minsky's "societies of mind". I make use of the Elixir programming language for its support for the Actor Model, and, more recently, an augmented Prolog to automate logical inferencing and constraint satisfaction. My efforts began before I became aware of Active Inference. The first iteration is based on an ad hoc model of cognition that makes no provision for learning. The second iteration is influenced by the "predictive brain". In this implementation, a society of actors make predictions, raise prediction errors, update beliefs and carry out action policies. Though it works, the model is conceptually muddled and learning is restricted to improving action policy selection. Furthermore, the implementation is riddled with "bottleneck" cognition actors. The third iteration takes a unified approach, with cognitive functions distributed over a society of generative model actors. This model of cognition is fully decentralized and aligns with the ontology of Active Inference. Learning, however, is still limited to policy selection. The current and fourth implementation, a work in progress, expands on this model and aims at significantly extending the scope of learning. Instead of a robot being given a fixed set of predefined generative models, it will grow and evolve them from experience and without human supervision. This will be attempted by dynamically instantiating generative models and having them use an "apperception engine" to induce, from their past perceptions[1] and actions, the logic programs that predict incoming perceptions, interpret perceptions into beliefs, and determine consequent action policies. These self-taught generative models will also abduce latent objects suspected of causing perceptions, and develop a shared vocabulary of predicates with which to express beliefs and predictions.

[1] In this paper, perception is synonymous with observation/sensation, with beliefs being derived from perceptions.

1. Roadmap

I am a software developer, not a neuroscientist. I am keenly interested in cognition, natural and artificial. Programming toy robots is my tool of choice to explore models of cognition and make otherwise abstract concepts tangible.

This paper describes an ongoing project to implement a succession of cognition process models on autonomous Lego rover robots. My ultimate goal is to program robots that conform[2] to the principles of Active Inference and learn as much as possible without human supervision.

Three models have been successfully implemented and a fourth one is just now entering development.

Version 1 predates my being aware of Active Inference and implements an ad hoc model of cognition as an instance of Minsky's Society of Mind[1] concept. Version 2 is the result of encountering the "predictive brain" theory and implements a "predictive society of mind". Version 3 is a more mature take, with generative models playing a central and unifying role in the robot's society of mind.

Learning is central to cognition. While each version led to reasonably competent robots, they all lacked meaningful autonomous learning capabilities, version 1 being essentially devoid of any.

Version 4 expands on version 3 and aims at endowing robots with greatly expanded learning capabilities, namely the ability to grow generative models and evolve their capabilities, without human supervision, from interactions with their environment. It is hoped that, if this effort is successful, an Active Inference analysis of the robots behaviors would demonstrate that they tend over time to minimize their Free Energy[3]. This is clearly ambitious. My expectations are that, short of success, interesting problems will be explored and intriguing questions raised.

2. Model v1 - A Society of Mind for Lego EV3 robots

In 2017 when I became aware that Lego EV3 robots could be programmed in Elixir[4], a high-level language that implements the Actor Model[2]. In the Actor Model, software is written as a collection of concurrent processes that manage their own data and communicate strictly via messages.

---

[2] By conforming, I mean found to minimize Free Energy when data collected from robot runs is subjected to analysis.
[3] How this would be conducted is to be determined.
[4] The Lego EV3 robotics toolkit allows Linux to run on its "computing brick". The ev3dev Linux distribution adds drivers that give read/write, file-based access to the EV3 sensors and actuators. This makes it easy to program an EV3 robot using any programming language Linux supports.

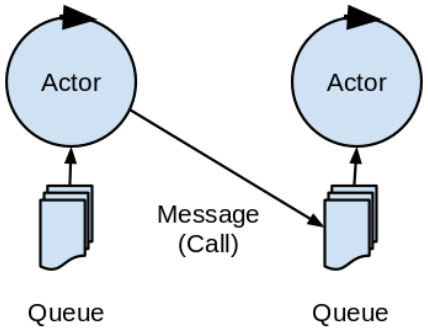


Fig. 1 - Actor Model

I had long been intrigued by a core hypothesis behind Marvin Minsky's Society of Mind, namely that seemingly intelligent behaviors could emerge from many simple, specialized processes interacting in simple ways.

Elixir processes are lightweight, allowing a large number of them to run fairly efficiently on the underpowered[5] Lego EV3's "computing brick". The EV3 brick connects to and controls a variety of sensors and actuators. Elixir running on the Lego EV3 looked like an excellent opportunity to test the Society of Mind hypothesis.

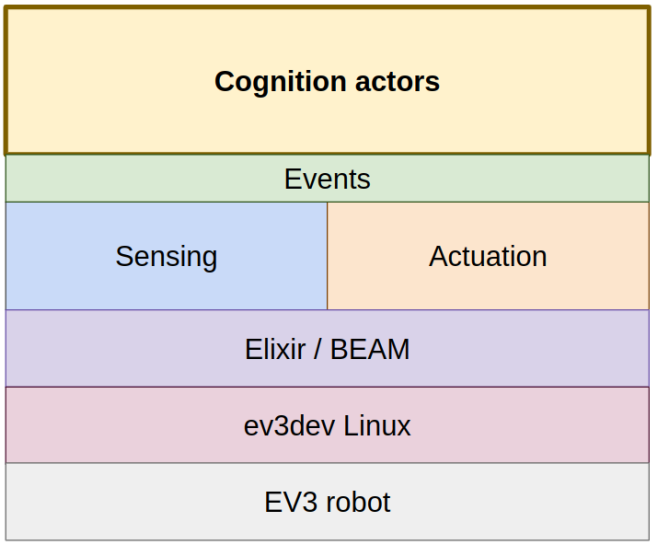


Fig 2. Elixir implementation

[5] Single core 300MHz processor with 64M of RAM.

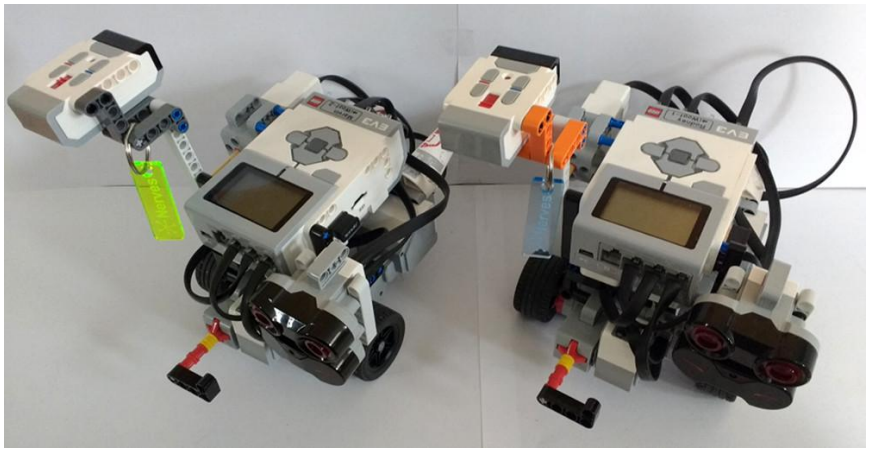


Fig. 3 Lego EV3 rover robots with EV3 brick

I improvised a model of cognition and implemented it as a collection of different kinds of “cognition actors" that exchange messages by publishing events that other actors subscribe to.

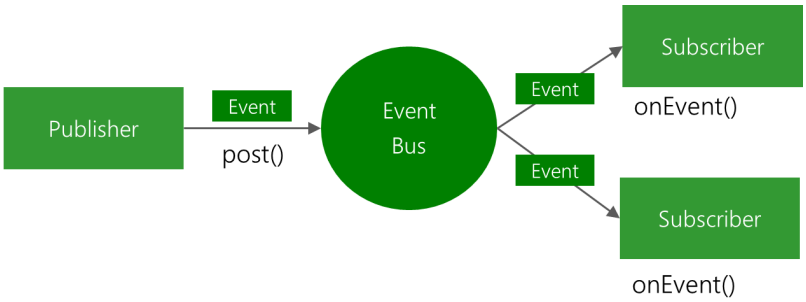


Fig. 4 Event publish and subscribe

In this initial model, detector actors periodically poll their associated sensors and publish “percept events". Perceptor actors listen to percepts of interest and integrate them into higher level percepts they then publish for other preceptors to process, and so on.

Motivator actors, each representing a different need of the robot (hunger, safety etc.), listen to percepts and publish “motive events”. Motives from higher-priority motivators silence motivators of lesser priority (safety > hunger).

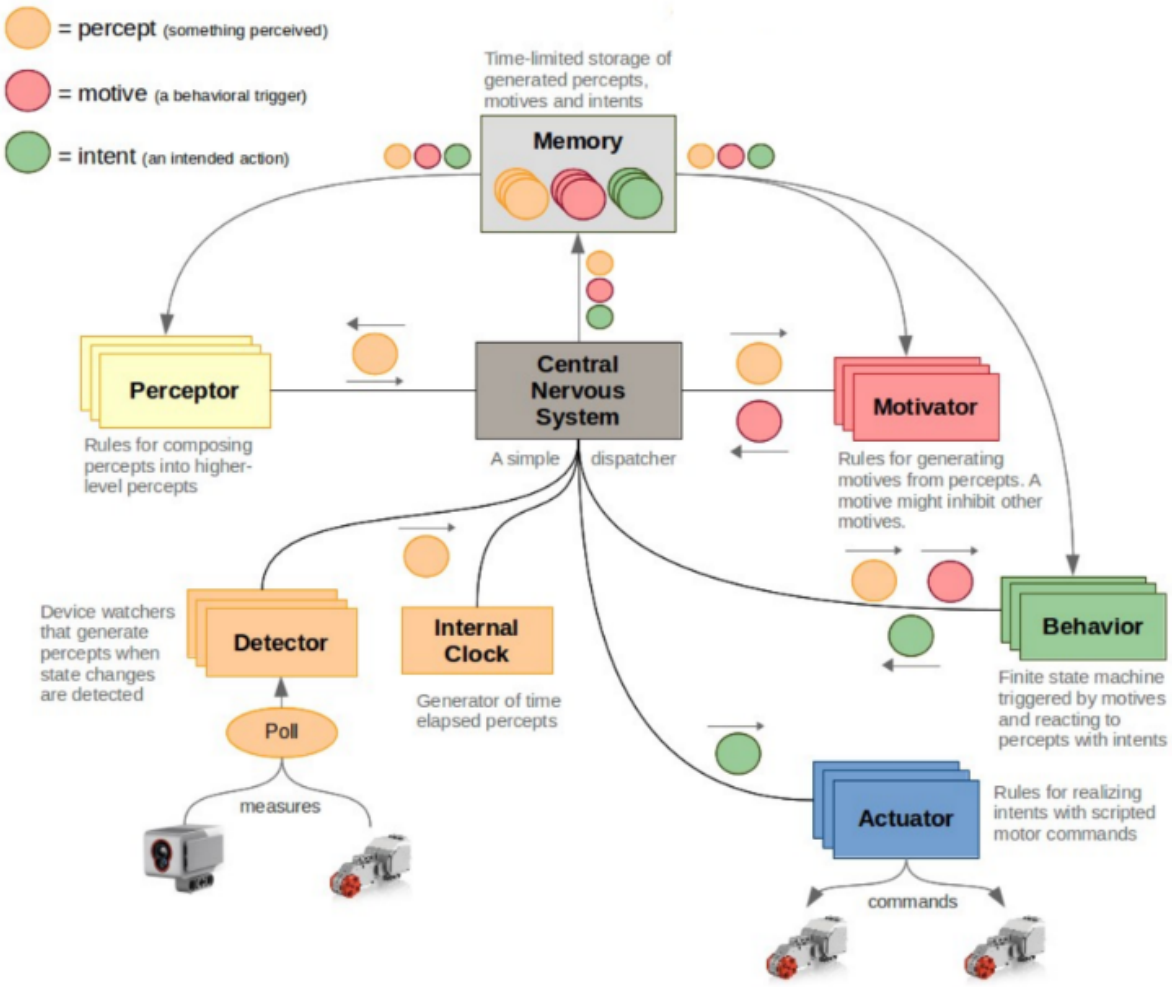


Fig 5. Cognition model v1

The motives that prevail trigger behavior actors. Behaviors are state machines that, once triggered, emit “intent events” whenever they transition from state to state . State transitions are caused by receiving new motives or percepts.

Some behaviors, labeled “reflexes”, are always active (they are not triggered by motives). Obstacle avoidance is realized by reflex behaviors.

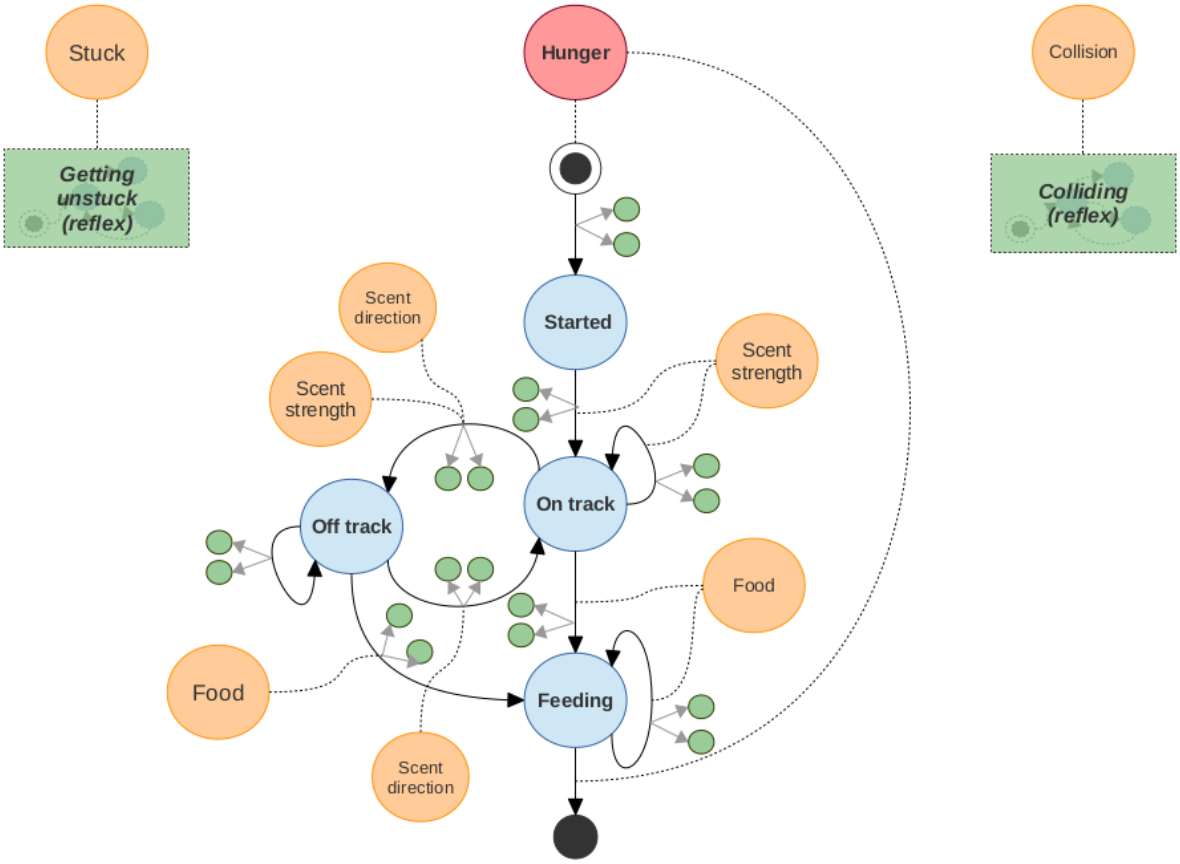


Fig. 6 Behavior state machine, cognition model v1

The intent events published by behaviors are listened to by actuator actors whose responsibility it is to translate intents into commands executed by the robot's motors.

Actors publish events to a "Central Nervous System" (a message bus) which then broadcasts them to subscribing actors. The CNS also communicates all events to a single Memory actor who keeps a timestamped record of them and makes them available for querying to actors (perceptors, motivators, behaviors) who might want to integrate past events in their decision-making.

The v1 model is successful inasmuch as robots do exhibit competent behaviors; they can avoid obstacles, get unstuck, seek food (color paper in front of an IR beacon simulating scent), and run around somewhat randomly when otherwise not preoccupied by hunger or their own safety.

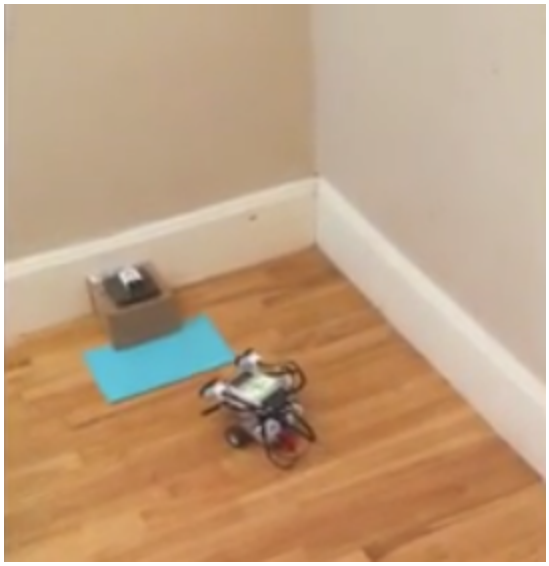


Fig 7. EV3 robot approaching food

However, the constant flood of percepts, motives and intents quickly overwhelms the limited processing capacity of the EV3 brick, with actors dealing with increasingly stale percepts, motives and intents. As a result, the robot finds itself reacting increasingly late to current events. The solution adopted is to detect when the robot is falling behind and to put it to sleep (disable actuators and detectors) and wait a few seconds[6] for events to "flush through the system" before waking up the robot. Clearly something to do with focus or attention is missing from this initial model.

Another issue is the absence of any learning capability. All preceptors, motivators and behaviors are predefined and their logic does not change as the robot interacts with its environment. All that changes is the flow of events, which motivators are inhibited, and which behaviors are triggered, and which intents are produced from the state transitions of active behaviors.

3. Model v2 - A predictive society of mind

Three issues from model v1 needed to be addressed:

1. The EV3 brick (the robot's CPU) is underpowered,
2. The lack of focus/attention causes a flood of percepts, most of which are not relevant at any given point in time,
3. Learning is absent.

The CPU problem is easily resolved by replacing the EV3 brick by a Raspberry PI 3[7] connected to a Dexter industries BrickPi3 board.

[6] This required the introduction of an "internal clock actor" generating "time tick events".
[7] Quad core 1.2 GHz CPU, 1 GIG RAM

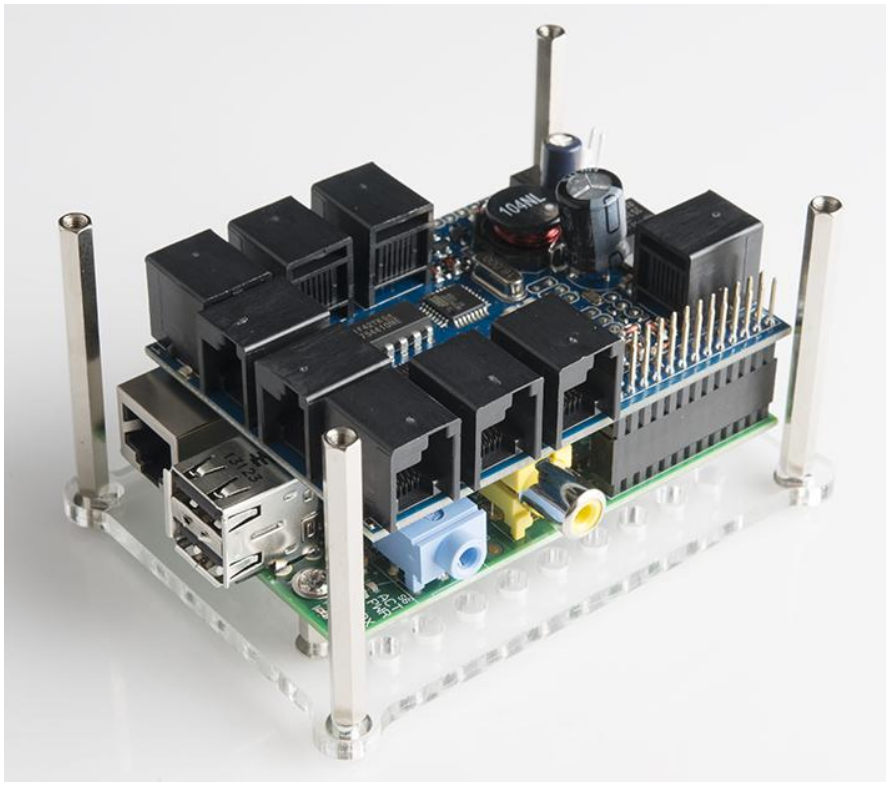


Fig.8 BrickPi3 board

The BrickPi3 provides the interface between the Pi3 and Lego sensors and actuators, affording greatly increased computing power to Lego EV3 robots.

One downside is that the BrickPi3 requires its own batteries, separate from the Raspberry Pi3's battery power source. This imposes significant bulk and weight requirements.

Fig. 9 Lego robot with Raspberry Pi3 and BrickPi3

The other issues (lack of focus, absence of learning) could not be addressed by tweaking the initial model. A model of cognition version 2 was called for. In 2018, I came across Andy Clark's book Surfing Uncertainty[3] about Predictive Processing. It offered an account of cognition that encompassed both focus/attention and had learning as a core function.

The Society of Mind model of v1 was overhauled into v2 by removing predictors, motivators, behaviors and replacing them with actors that implement predictive processing.

In this model v2, a robot given a profile that defines conjectures (the kinds of beliefs it can hold), predictions it can make to verify such beliefs, and fulfillment options for making predictions come true.

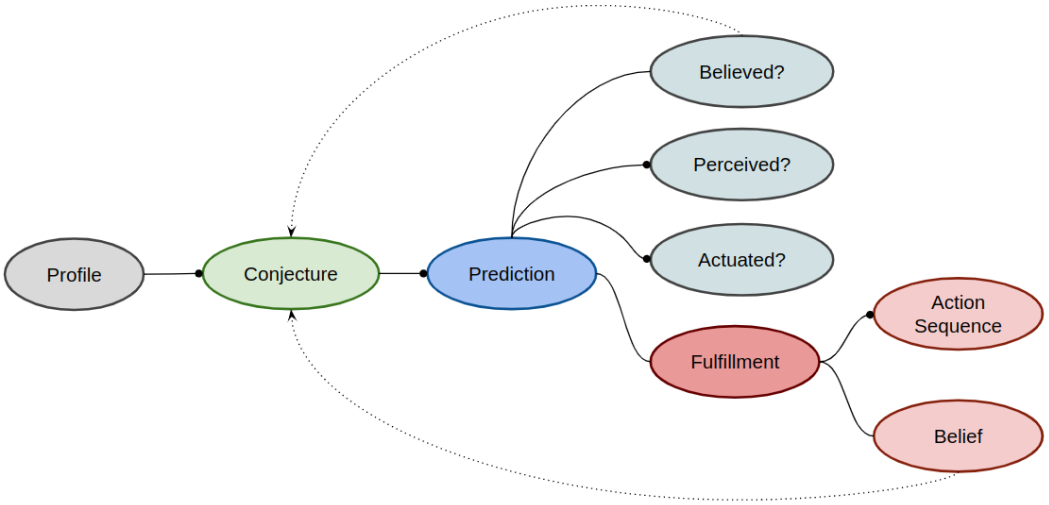


Fig 10. Profile concepts of cognition model v2

A prediction here is either an assertion about a belief being held, current perceptions or prior actuations. A fulfillment defines options for fulfilling a prediction. Options can be either alternate sequences of actions, or another conjecture to believe in.

For each conjecture in a robot's profile, a "believer actor" is instantiated whose sole purpose is to verify the conjectured belief. It does so by instantiating "validator actors" for each prediction by which the belief is verified. The validator executes logic to check if its assigned prediction is valid and, if not, raises prediction errors and attempts to fulfill it. Successful prediction fulfillment raises a "fulfillment event".

A unique "experience actor" processes prediction error and fulfillment events and keeps track of which fulfillment options correlate to success. It uses this information to recommend fulfillment options to validators on demand.

Focus and attention is realized by two eponymous actors. The focus actor enables and disables validators to ensure that only beliefs in the more important conjectures are the ones being "worked on" at any given time.

The attention actor is responsible for starting/stopping detectors so that only those of interest to the active validators are in play.

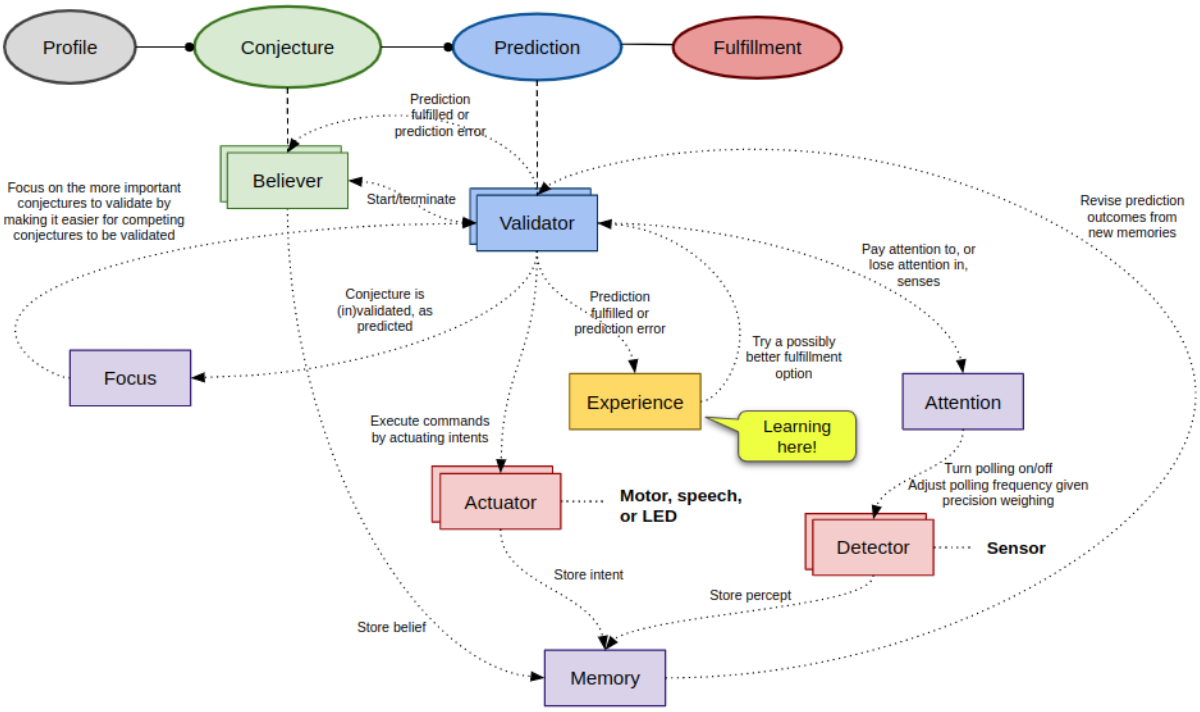


Fig. 11 Cognition model v2

Robots competently seek food, avoid obstacles and roam around under a regime of predictions, beliefs and prediction errors. They become noticeably better over time at selecting sequences of actions to fulfill predictions.

Yet the model is unsatisfactory, especially the interplay of beliefs, predictions and prediction errors which, on closer reading, does not quite align with the theory. Learning is also limited to action policy selection and the process model contains many singleton actors (experience, focus, attention, memory) with the attendant and unwanted computing bottlenecks.

4. Model v3 - A society of generative models

From a closer reading of Predictive Processing, I noted the central role generative models play in predictive processing and revised the domain model to make generative model actors the core components of a redesigned process model.

4.1 Domain model

An effort was also made to decouple at the implementation level the physical configuration of a robot (the platform) and its assigned generative models (the profile).

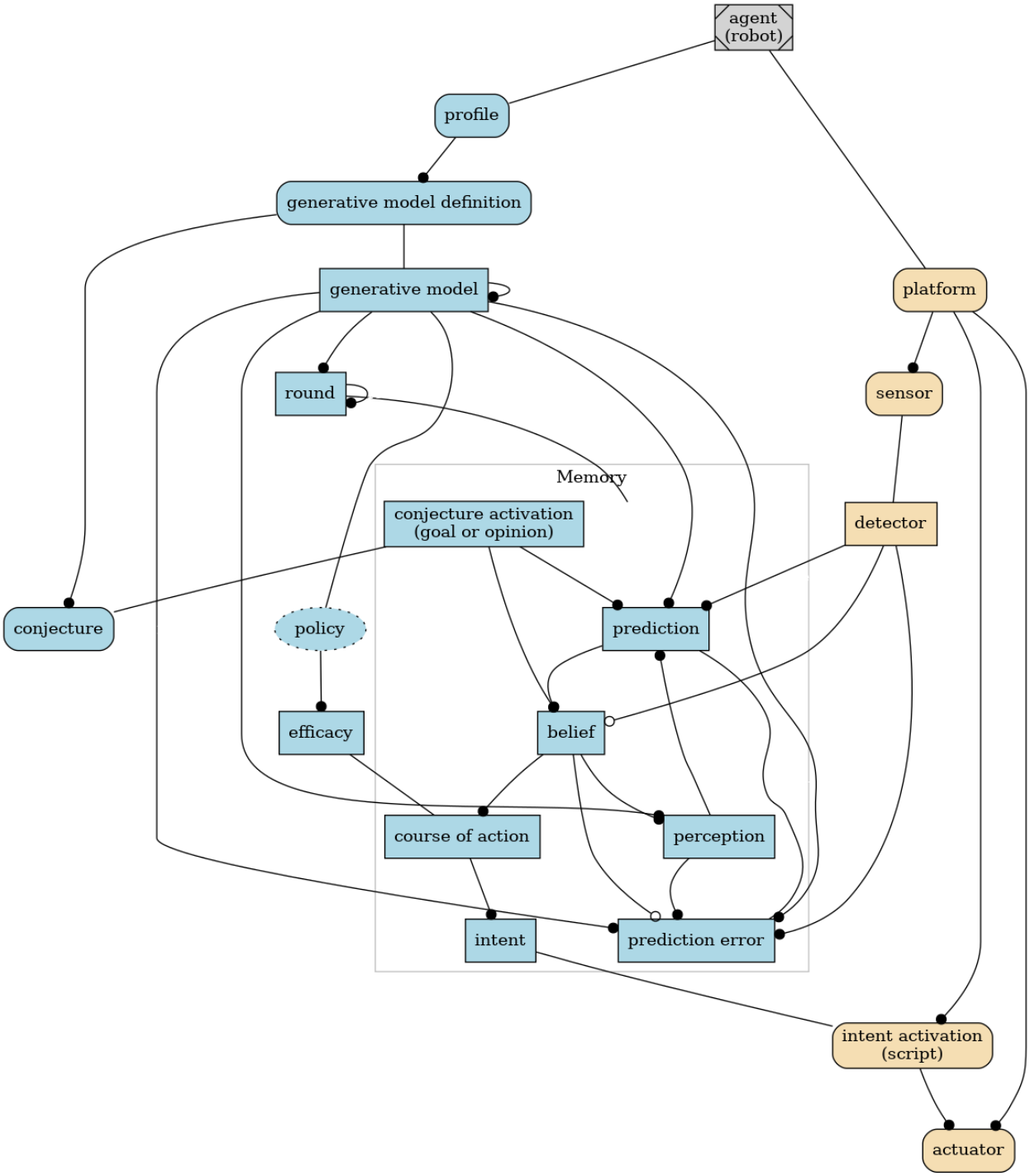


Fig. 12 Concepts map for cognition model v3

A *platform* describes the physical configuration of the *agent*, namely, its *sensors* and *actuators*. An *agent* (robot) is assigned a *profile* and a *platform*, from an inventory of predefined *profiles* and *platforms*.

The platform maps *intents* (e.g. going forward, turning left etc.) to coded functions that activate *actuators* to fulfill these *intents* given a specific morphology (e.g. to move forward, activate both wheels with positive polarity).

A *profile* lists *generative model definitions* (GM def). For each definition, a *generative model* (GM) is instantiated. A *profile* organizes these *GMs* into parent-child relationships.

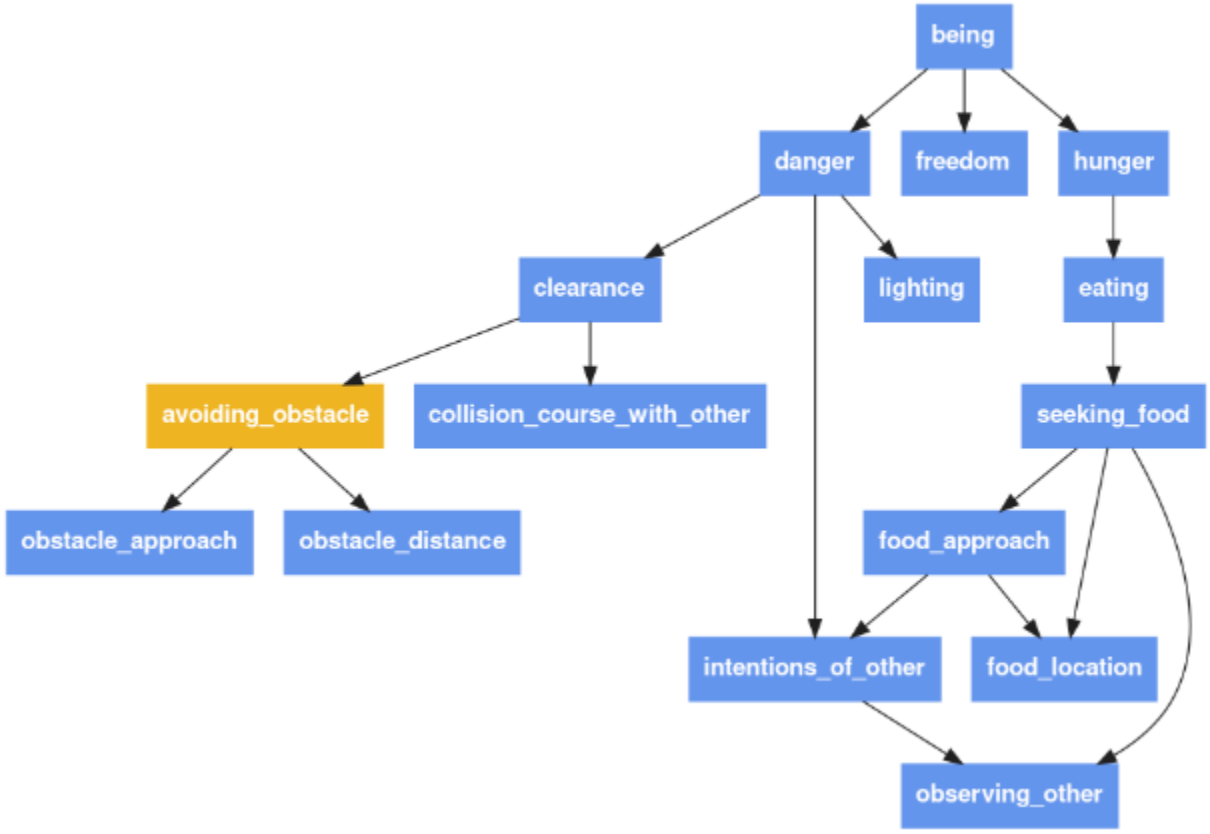


Fig. 13 Implemented hierarchy of generative models

Two robots could share the same *profile* but have different morphologies thus different *platforms*. Conversely, two robots could be physically identical (same *platform*) but have different *profiles*, and thus each interact differently with their environment.

A *detector* is instantiated for each of the platform's *sensors*. A *detector* is akin to a trivial generative model whose *beliefs* are exactly the *sensor's* inputs. A *detector* receives *predictions* from *GMs* but does not produce any. A *detector* raises *prediction errors* when its inputs contradicts *predictions* it receives. *Detectors* sit implicitly at the bottom of the *GM* hierarchy.

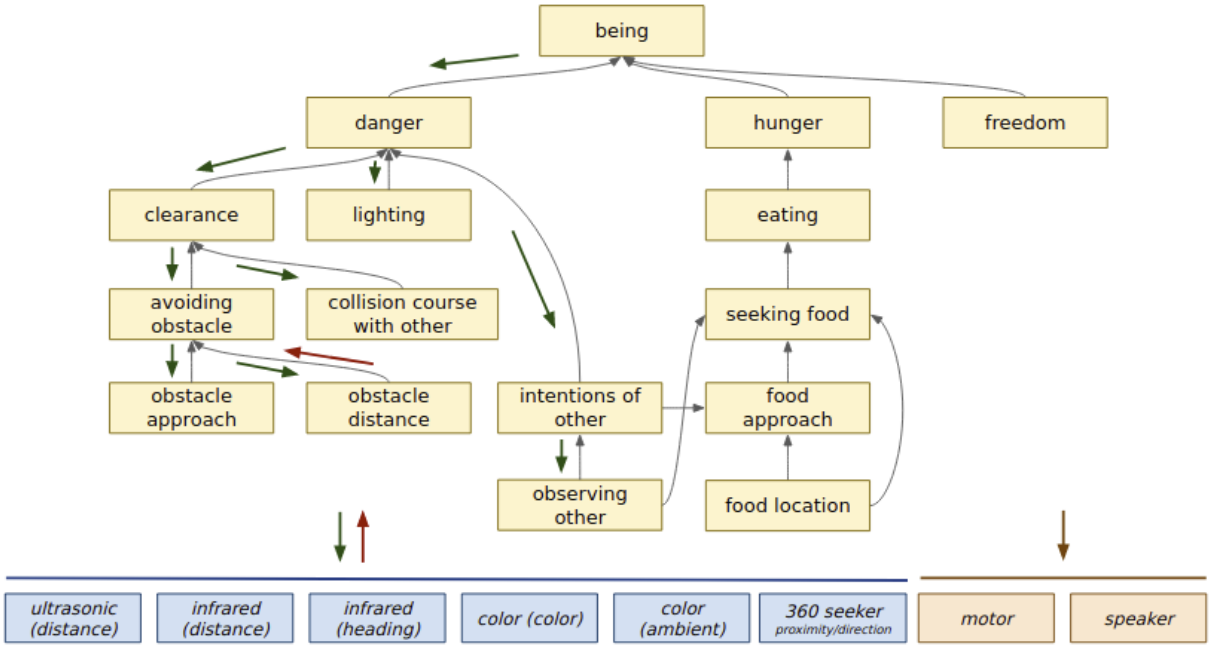


Fig. 14 Exchanges of predictions and prediction errors

A *GM* definition establishes a set of *conjectures* for a *GM*.

A *conjecture* names a *belief* (categorized as a goal or opinion), sets how to update it from *perceptions*, and how to raise *predictions* should the *belief* be held. The logic is defined as parameterized software functions.

A *GM's conjecture* is activated (*conjecture activation* executes) when the *GM* receives a *prediction* from another *GM* about the *conjecture's belief*.

4.2 The lifecycle of a generative model actor

A *GM* operates in a sequence of *rounds*, one *round* at a time. A *round* is time-boxed.

During a *round*, a *GM* may receive *predictions* from other *GMs* about its own *beliefs*. This activates related *conjectures* which, in turn, raises *predictions* about child *GM beliefs*.
Within a *round*, the GM's *perceptions* consists of the list of *prediction errors* it has received, plus the *predictions* it has raised for which no *prediction errors* were received with the round.

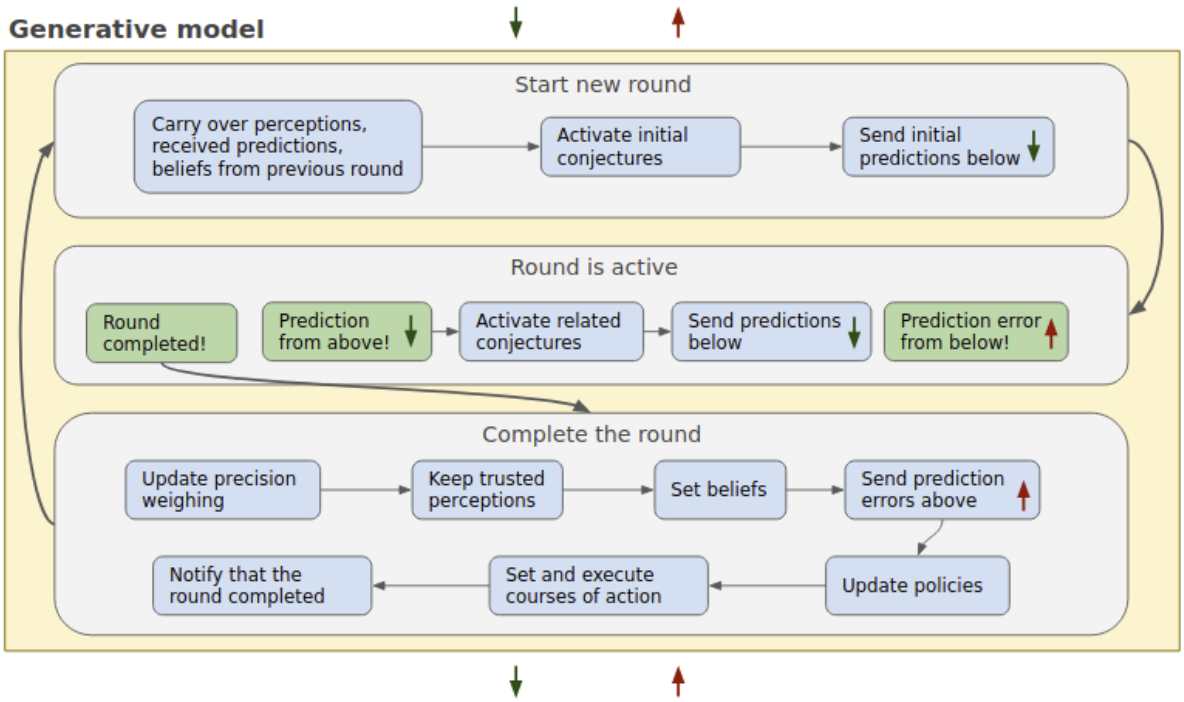


Fig. 15 Lifecycle of a generative model actor

During the execution of the *round*, the *GM* may receive *prediction errors* from child *GMs*, integrate them into its perceptions, update its own *beliefs*, and possibly raise *prediction errors* if these *beliefs* contradict the *predictions* it received during the *round*.

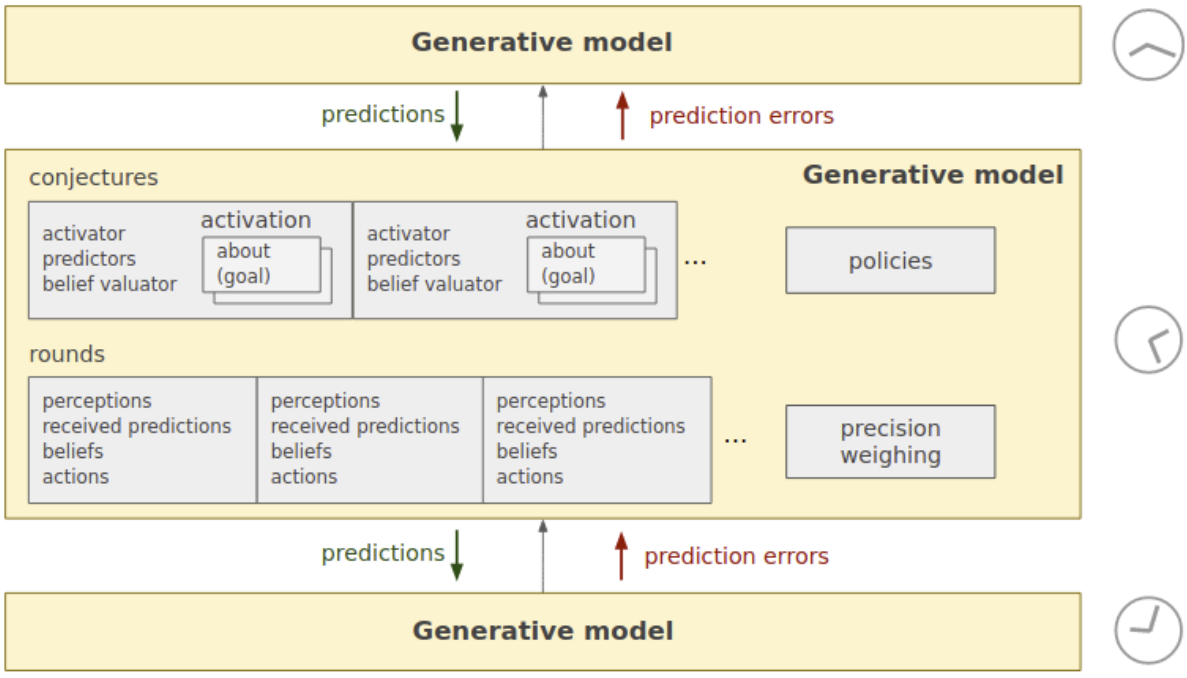


Fig. 16 Interactions between generative model actors

A *GM* updates its *beliefs* (using functions contained in its *conjectures*) at the end of a *round* from its current *perceptions*.

After having updated its *beliefs*, a *GM* determines *courses of action* most likely to achieve each goal *belief,* or to validate opinion *beliefs* it currently holds.

A *course of action* is a sequence of *intents*. The *GM* favors *courses of actions* that correlate with high *efficacy*. However the *GM* will also sometimes attempt new *courses of action* or prior, less optimal ones to avoid getting stuck in a local minimum.

Computed "*efficacies*" make up the learned policy know-how of the *agent*. They are remembered across runs of the agent.

The goal *beliefs* of a *GM* are carried over from *round* to *round* until they are achieved, or until the *GM* stops receiving *predictions* about these goal *beliefs*, signaling loss of attention.

A *GM* keeps a memory of recent *rounds*. For each past *round*, it remembers *perceptions*, *predictions* produced, *prediction errors* raised, *beliefs* held, *courses of action* taken. This memory provides a temporal context to the current *round* to compute the latest *beliefs* and *predictions*.

4.3 Building a simple theory of mind

A 360 degree modulated infrared beacon and wide-angle IR detector were added to each robot’s platform (body) so that one robot could detect another robot’s relative position via its infrared signature.

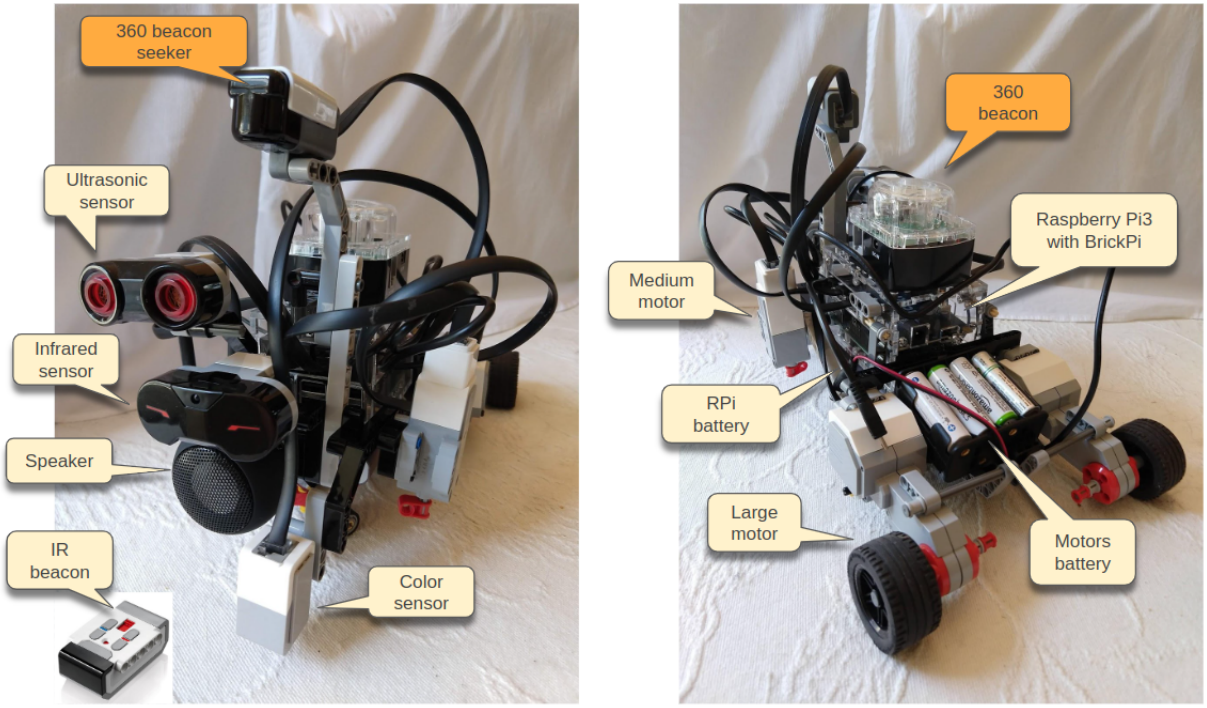


Fig. 17 Lego robots with identifying infrared beacons and IR sensors

This ability of a robot to perceive and remember the relative positions of another robot makes it possible to add generative models whose concern is to infer the intents of another robot. This initial attempt at constructing a “theory of mind” is limited to detecting properties of another robot’s recent trajectory and inferring intent from it.

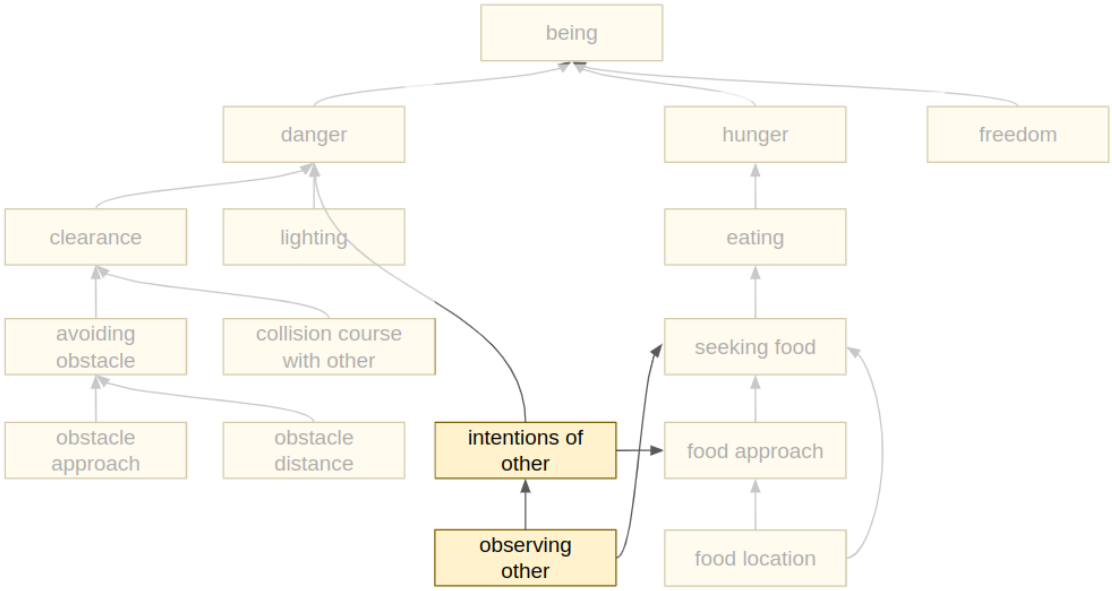
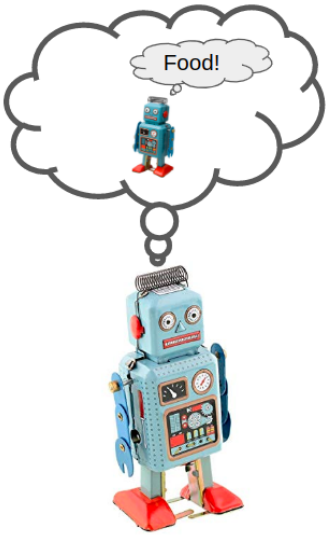
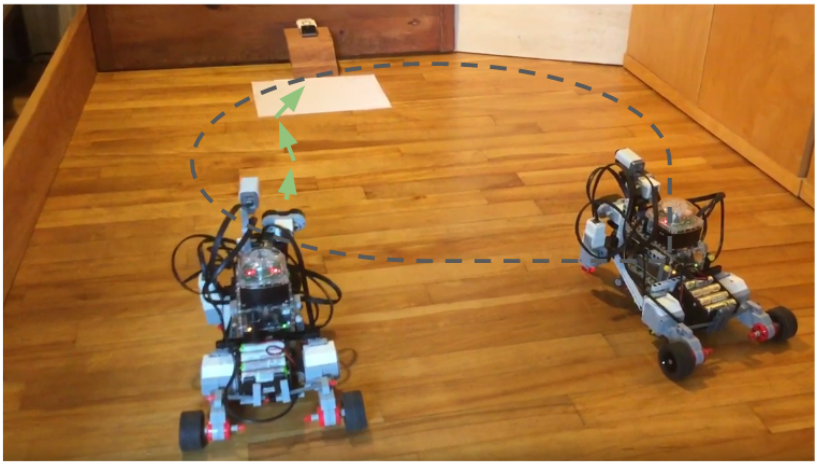


Fig. 18 Theory of mind generative models

If a robot perceives another to be “taking a bee line”, it infers that the other robot is tracking a food source and may follow the other robot.





Other rover homing on food is inferred from “beeline” trajectory

Fig. 19 Inferring feeding intent

If the other robot is perceived as moving chaotically, then it is inferred to be “in a panic” (the “danger” generative model produces random sequences of action) and the witnessing robot reacts by also believing itself in danger[8].

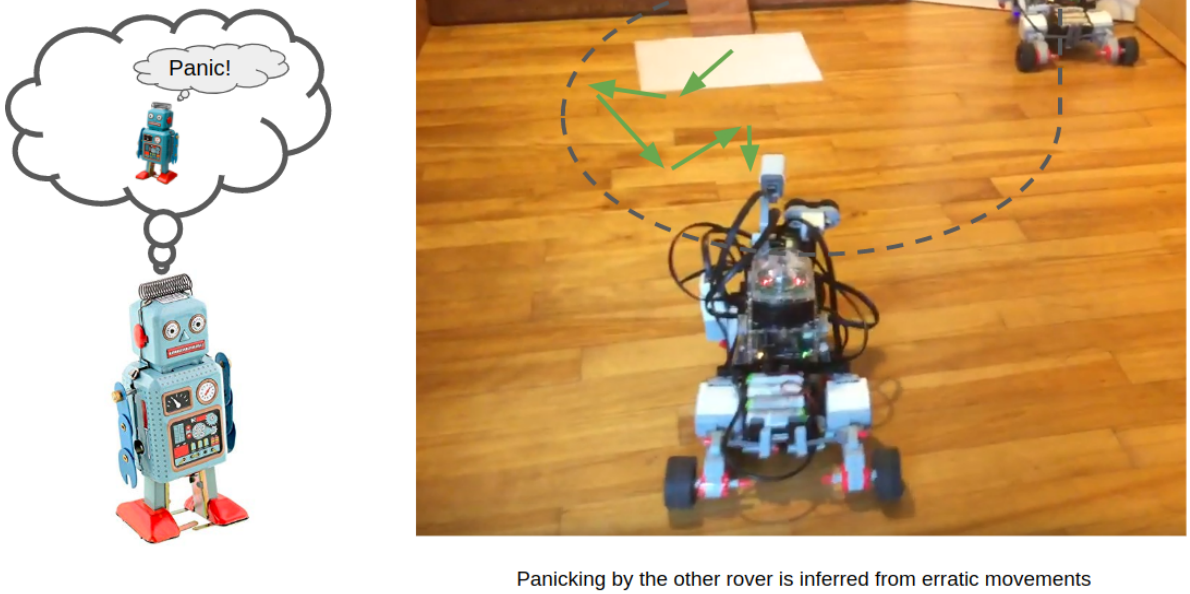


Fig. 20 Inferring panic intent

Debugging software can be difficult in the best of circumstances. Debugging a highly concurrent, actor-based process model running a rich hierarchy of generative models on a robot in real time is especially difficult.

4.4 Simulating robots

To accelerate debugging and testing, virtual robots can run in a simulation environment. Changes in the internal states of virtual robots can be observed, slowed down, paused and replayed.

[8] Provisions are made in the belief updating logic of the danger GM to avoid a vicious cycle of panics.

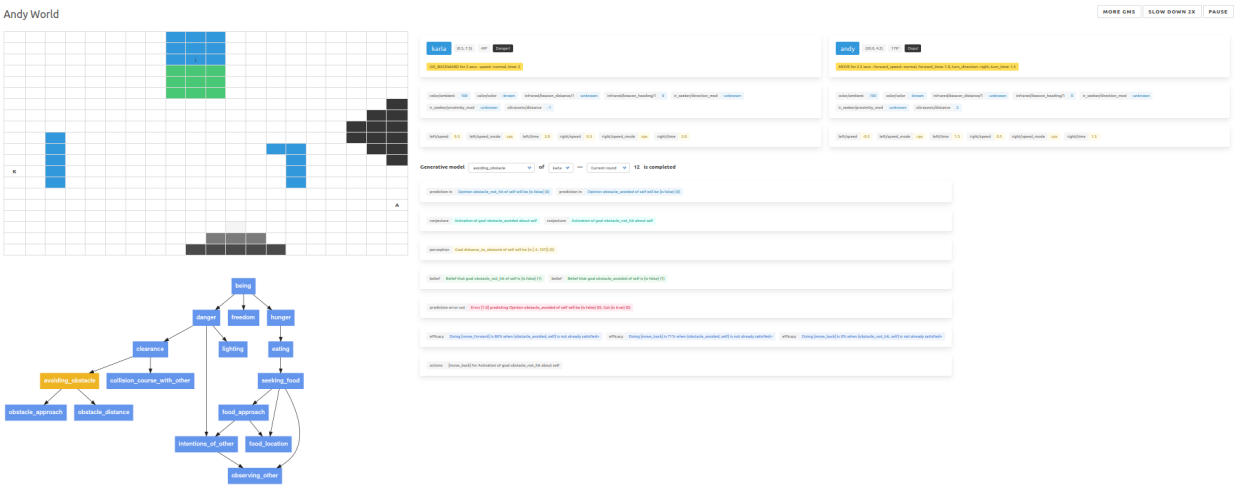


Fig. 21 - Simulation environment

The simulation environment puts virtual robots in a grid world with obstacles (the sides of the grid and blue squares), food (the green squares) and more or less dark areas (black and gray squares). The robots are represented by the first letter of their names.

When in simulation mode, the implementation runs on a personal computer instead of on a robot. Detector actors get their perceptions from the virtual robot's sensors. Actuator actors translate intents into commands to virtual motors, moving the virtual robot in the grid world and changing what it perceives.

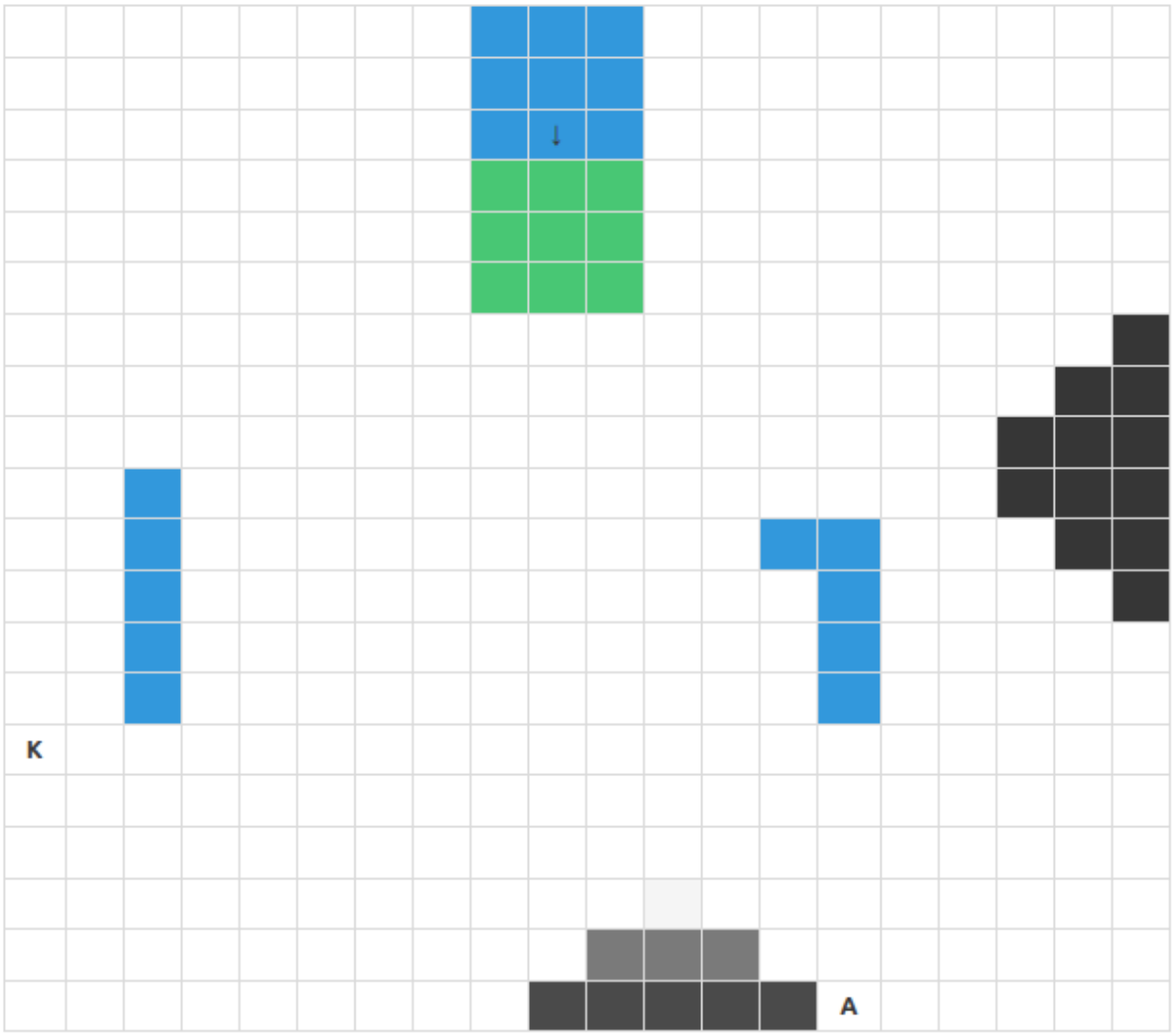


Fig. 22 - Simulation environment grid world

A robot's generative model can be selected and observed as the robot goes through its paces. One can see the predictions being received, conjectures being activated and beliefs updated, prediction errors raised and received, and actions being intended and taken.

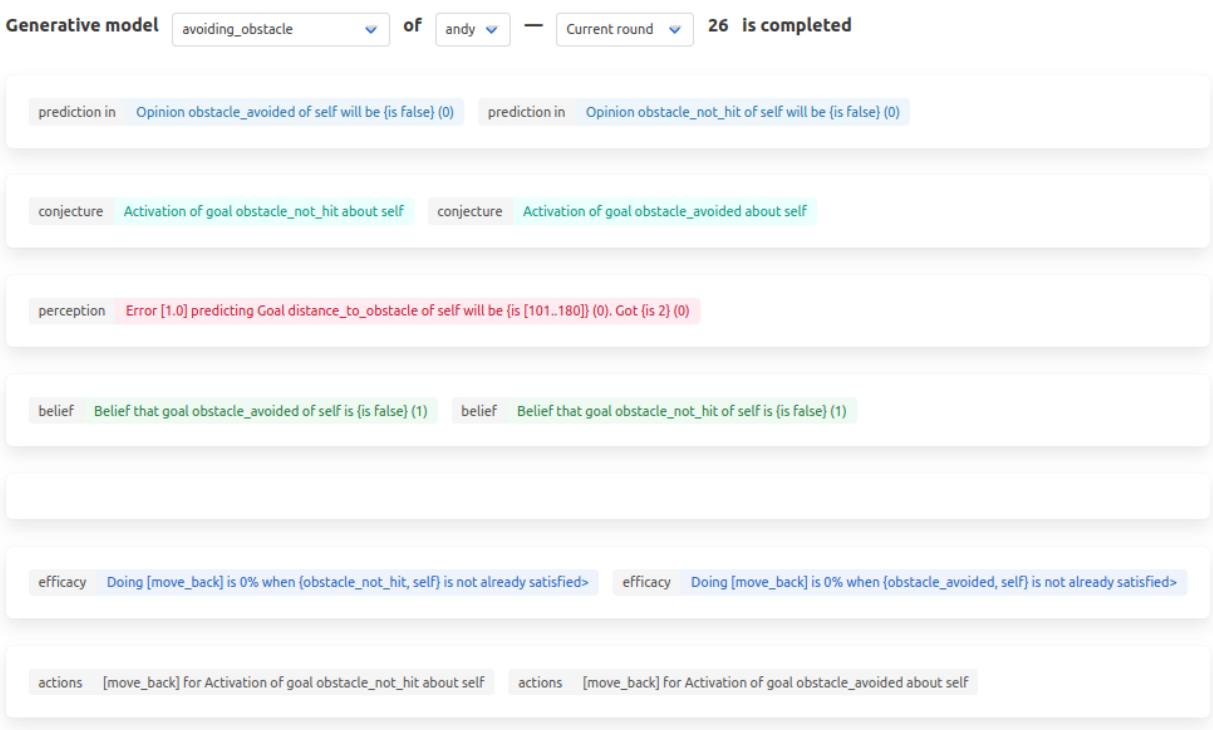


Fig. 23 Generative model dashboard

The current position and orientation of each robot is displayed, as well and the current state of its actuators.

Fig. 24 Robot state dashboard

4.5 Model evaluation

The generative model-centric model (v3) conforms quite well with the ontology of Active Inference[4] and thus can arguably be described as a “symbolic implementation of Active Inference”, though an incomplete one.

It is incomplete mostly in that learning is still limited. Model v3, just like model v2, restricts learning to policy selection (by correlating actions to successful belief updating). However, model v3, by integrating all predictive processing functions into generative models, sets the stage for a new model that greatly expands the learning capabilities of the robots.

5. Model v4 - Active inferencing

The latest version is in early development. Its scope is ambitious: A robot is to discover and evolve (i.e. learn) its generative models from interactions with its environment without human supervision.

5.1 Prior capabilities

The implementation will provide a robot with a minimum of prior capabilities, such as the generic machinery of generative models (a GM framework) but not their idiosyncratic logic; this will be synthesized by the apperception engine.

A robot will aso “start life” with all the detectors and actuators needed to interface with the robot’s sensors, motors etc. It will possess a fully-defined “metacognition generative model” responsible for seeding new generative models and weeding out dysfunctional ones[9].

Finally the implementation will include an “apperception engine” to generate logic programs a generative model uses to carry out its predictions, belief updating and policy selection.

From Richard Evans et al.’s Making sense of sensory input[5]:

> *This paper makes two main contributions. The first is a formalization of what it means to “make sense” of the stream of sensory data. According to our definition, making sense of a sensory sequence involves positing a symbolic causal theory – a set of objects, a set of concepts, a set of initial conditions, a set of rules, and a set of constraints – that together satisfy two conditions. First, the theory must explain the sensory readings it is given. Second, the theory must satisfy a particular type of unity. Our definition of unity involves four conditions. (i) Spatial unity: all objects must be unified in space via a chain of binary relations. (ii) Conceptual unity: all concepts must be unified via constraints. (iii) Static unity: all propositions that are true at the same time must jointly satisfy the set of constraints. (iv) Temporal unity: all the states must be unified into a sequence by causal rules. Our second contribution is a description of a particular computer system, the Apperception Engine , that was designed to satisfy the conditions described above.*

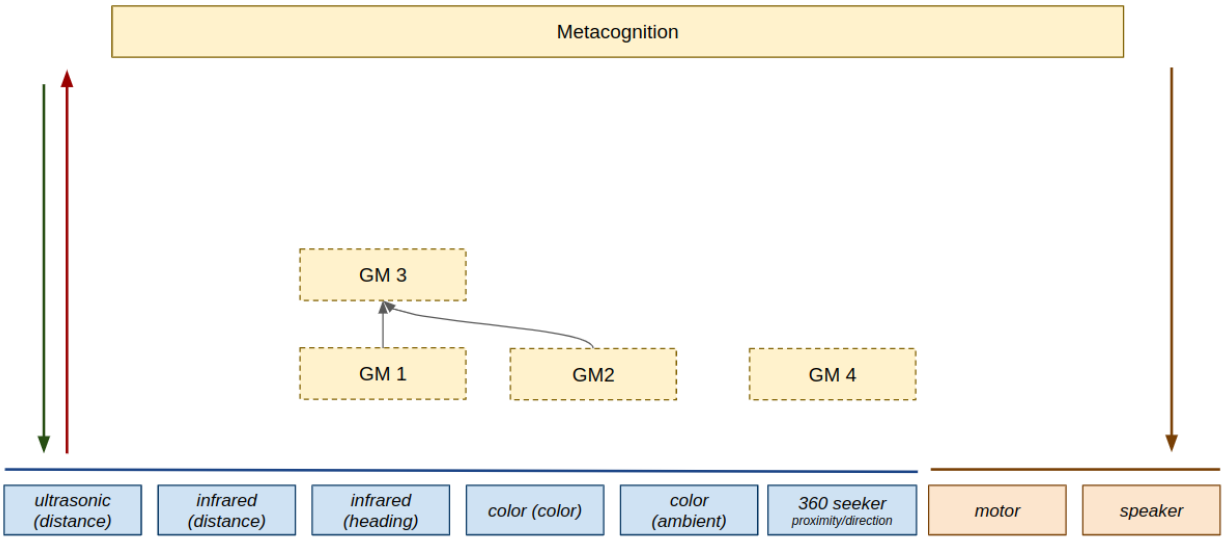


Fig. 25 Structure learning

[9] Meta-learning is out of scope.

5.2 The apperception engine

Once a generative model is instantiated, it employs the apperception engine to determine or adjust its scope, i.e what its predictions and beliefs can be about, and to generate the logic by which it makes predictions, updates its beliefs, and formulates action policies.

The apperception engine searches a space of candidate logic programs, informed by a GM's past rounds and the scopes of other GMs it builds upon and build upon it. The search is subject to semantic and time constraints.

The task of the apperception engine is to efficiently generate competent code from past experience. To synthesize prediction-making logic, i.e. the GM's predictor program, the apperception engine takes the perceptions from the GM's past rounds and searches for a logic program (deduction rules) that, given the perceptions of a given round, deduces the perceptions of the subsequent round (possibly a superset of them).

A predictor needs a defined scope that constrains what objects the predictions are about (self, food, obstacles, etc.) and what predicates can be used to express these predictions (distance-of, touches, orientation-of, etc.)

The scope of a GM mostly borrows from the scopes of other GMs and of detectors. After all, a GM makes predictions about the beliefs of other GMs and/or detectors. A GM might decide to abduce (imagine) other objects not contained in the scopes of others. The scope of a GM determines if the GM is more or less abstract than another. A GM that expands the scope of another is considered "higher level" or more abstract.

There are three kinds of synthetic logic rules of a predictor:

1. Static rules that, together, constrain the perceptions in a round (mutual exclusion rules) and that infer implied perceptions.
2. Causal rules that determine perceptions in a round given the known and implied perceptions in the previous round.
3. Unity (semantic) rules that constrain all predicates and objects used in the above rules to be conceptually connected: An object referenced by a rule must be connected via rules to every other referenced object. Every predicate must be universally quantified or related via mutual exclusion to other predicates.

The space of all possible such logic programs is extremely large. Constraints must be applied to restrict the search space so that an acceptable candidate can be found within a reasonable amount of time.

The perceptions in the sequence of rounds a GM remembers constrain the search for a predictor. Valid predictors must infer (a superset of the) perceptions in round N+1 given those in round N. Unity rules further restrict the set of valid predictor programs.

Heuristics also come into play to further restrict the search space and maximize the chances of finding an acceptable candidate in a limited amount of time, heuristics such as:

- Start search from small scopes
- A GM's scope must always be identical to, or a strict superset of, the scope of another GM
- Predictors with the fewer and least complex rules are preferred.

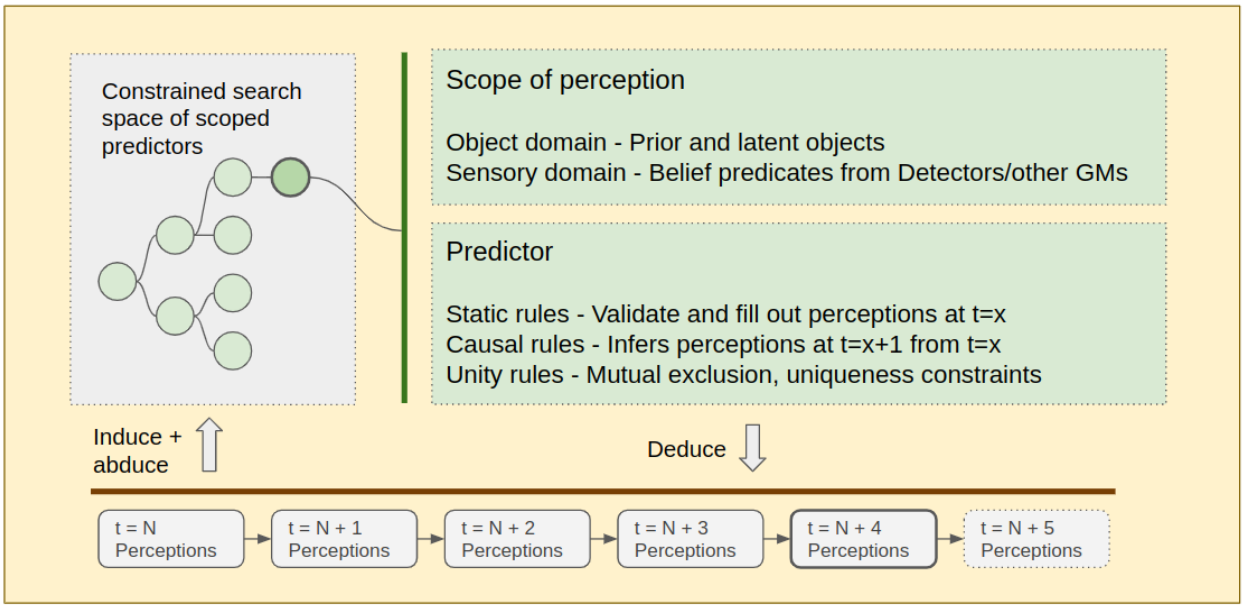


Fig. 26 Generating predictors with an apperception engine

Similar constraints apply to the synthesis of belief updating and policy selection logic programs.

5.2 Homeostasis and feelings

There needs to be a "value function" to guide the process of growing and evolving a "society of generative models". Homeostasis plays this role. The GMs evolve in the right direction if the robot's homeostasis is generally less at risk over time.

The "homeostatic score" of a robot is an aggregate value derived from the emotional valence of all beliefs currently held by the robot's GMs. A belief has emotional valence if it is derived, directly or indirectly, from perceptions of positive or negative "feelings".

Feelings are interoceptive sensations generated by predefined code functions that evaluate deviation from homeostasis.

Feelings have a valence that is either negative, neutral or positive.

The robot can feel:

- Hunger - negative if internal energy is depleted due to action and computation
- Pain - from bumping into things etc.
- Ennui - negative if it has not been learning for an extended period of time

Each kind of Feeling has an assigned charge that multiplies its valence, allowing a Feeling to dominate another. Emotional charges can vary over the life of an agent, for example downplaying pain and hunger in favor of dispelling ennui.

Maximizing the emotional valence of beliefs held will prioritize action policies selected by GMs over others if they address more emotionally charged beliefs, Variations in Feelings over time will be used to evaluate the fitness of learned capabilities.

A capable agent will be one that, through its actions, ends up "feeling good" more often than not.

Metacognition

Guiding a society of generative models up an "homeostatic gradient" (or keeping it from backsliding) is the responsibility of the Metacognition GM.

The Metacognition GM sees the GMs as "cognition sensors".

Each GM produces a stream of "cognitive sensations" consumed by the Metacognition GM, namely:

- A predictive success measure (Are most of my predictions contradicted?)
- A learning measure (Am I finding better predictors etc.?)
- An emotional measure (Am I feeling mostly good or bad?)
- An effectiveness measure (Are my policies effective?)

The Metacognition GM perceives the cognitive sensations of GMs and the Feelings of the robot. It combines these perceptions into beliefs about the individual and joint capabilities of the GMs. It may then decide to carry out policies that add and cull GMs so as to improve the agent's overall capabilities.

5.3 Implementation

A common thread unites the implementation of all four models, the Society of Mind concept whereby a robot's visible behaviors emerge from "actors" of various kinds concurrently producing and reacting to events.

So far, Elixir, with its best-in-class support for the Actor Model, has served all implementation needs well. With the expansion of learning capabilities requiring an apperception engine to synthesize logic programs, this is no longer entirely true.

An apperception engine is best implemented in a logic programming language capable of constraint solving such as SWI-Prolog augmented by CHR (Constraint Handling Rules). Prolog will also provide the execution environment for the logic programs synthesized by the apperception engine.

```
% A predictor for a GM given a sequence of per-round perceptions (sensory sequence)      You, 1 second ago • Uncommitted changes
predictor_for_sequence(GM, Sequence, Predictor) :-
    % Choose a sub-sequence, prioritizing on recency then on length
    sub_sequence(SubSequence, Sequence),
    % Guess an object and sensory domain that covers the sub-sequence
    prediction_scope(GM, SubSequence, Scope),
    % List typed variables given the GM's scope
    variables(Scope, Variables),
    % Guess unity rules
    unity_rules(Scope, Variables, UnityRules),
    % Conceptual unity requirement met? (Each predicate in scope mentioned in a unity rule)
    conceptually_unified(Scope, UnityRules),
    % Guess static rules
    static_rules(Scope, Variables, StaticRules),
    % Guess causal rules
    causal_rules(Scope, Variables, CausalRules),
    % Guess initial conditions from the sub-sequence
    initial_conditions(SubSequence, InitialConditions),
    % Compute the trace (if computed, it implicitly meets static and temporal unity requirements)
    sensory_trace(InitialConditions, UnityRules, StaticRules, CausalRules, Trace),
    % Verify spatial unity requirement met (for each pair or objects in scope, there's a direct or indirect relation connecting them.)
    spatially_unified(Scope, Trace),
    % Verify that the Trace subsumes the sub-sequence
    sensory_trace_covers(Trace, SubSequence),
    % Assemble a new candidate predictor!
    length(SubSequence, Coverage),
    Predictor = predictor{
        coverage:Coverage,
        scope:Scope,
        initial_conditions: InitialConditions,
        variables:Variables,
        unity_rules:UnityRules,
        static_rules:StaticRules,
        causal_rules:CausalRules}.
```

Fig. 27 Sample code from an apperception engine's Prolog implementation

Because of the close relationship between generative model actors and the apperception engine, it would be counterproductive to keep implementing GMs in Elixir while the implementation of how and what they learn is in Prolog.

Adding an actor and eventing extension to SWI Prolog enables migrating GM actors to Prolog while leaving sensing and actuation in Elixir. This allows a system architecture that "cuts at natural joints", with embodiment (actual and virtual) implemented in Elixir and active inferencing implemented in Prolog.

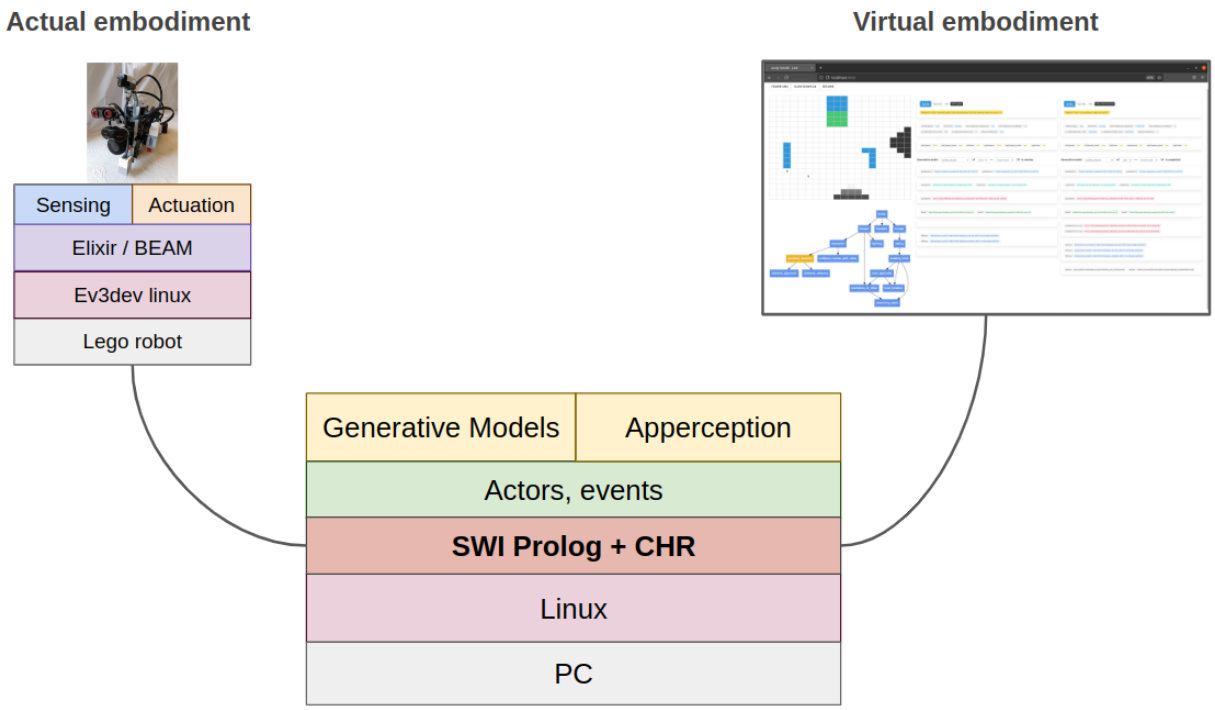


Fig. 28 Implementation decoupling cognition actors (Prolog) and embodiment actors (Elixir)

6. Concluding remarks

All three completed iterations, each with its own model of cognition and each implemented as a society of actors, were successful in that they produced reasonably competent behaviors on roving Lego robots. The robots all managed to track food, while avoiding obstacles and, sometimes, each other. In one model, a robot could even infer, in a very limited way, the intents of another robot.

The first model was ad hoc while the others increasingly aligned to the concepts and principles of Active Inference.

The current iteration has ambitious goals, not least among them:

- The implementation of an apperception engine that successfully and efficiently generates code that, when run, predicts perceptions, updates beliefs and selects action policies for a GM.
- Measuring on a robot deviation to homeostasis and expressing it as feelings that give an emotional charge to beliefs, and using this to prioritize possibly conflicting action policies.
- Implementing metacognition capabilities that guide the evolution of a society of generative models along a homeostatic gradient, without micromanaging the process.

This effort has already raised questions that, I hope, will be explored if not answered:

- How much does growing a society of generative models have in common with developmental biology?

  For one thing, the number of GMs starts small (one) and grows. Each new GM shares the same “genetic code” (its framework code) yet differentiates. Furthermore, the “phenotype” of a GM (its scope and synthesized logic programs) is constrained by “neighboring” GMs (those it depends on for its perceptions).

- What is the “minimum viable priors” a robot needs to engage successfully in autonomous learning?

  Some functionality must be present apriori without which autonomous learning necessarily fails. For example, some predefined constraints/heuristics are essential to pruning the search tree of the apperception engine. Without them, acceptable solutions are unlikely to be found in a reasonable amount of time.

- Under what conditions does unsupervised learning tend to converge on competency? Diverge?

  As a robot learns it engages in “babbling”, i.e. trying out different logic programs for obtaining perceptions, updating beliefs, and selecting actions, all with different degrees of success. Babbling is useful if it leads to better programs and thus greater competency at preserving homeostasis. Babbling is useless if it does not. Distinguishing useful from useless babbling matters.

Acknowledgements

This work was supported in part by a grant from the Maine Technology Institute and by SmartRent. I also wish to thank the members of the Active Inference Institute who provided much appreciated support and feedback.

References

[1] Minsky, Marvin (1986). The Society of Mind. New York: Simon & Schuster. ISBN 0-671-60740-5.
[2] Hewitt, Carl (2015). Actor Model of Computation: Scalable Robust Information Systems, arXiv:1008.1459 [cs.PL].
[3] Clark, Andy (2016). Surfing Uncertainty: Prediction, Action, and the Embodied Mind. Oxford University Press. ISBN-13: 9780190217013.
[4] Thomas Parr, Giovanni Pezzulo and Karl J. Friston (2022). Active Inference - The Free Energy Principle in Mind, Brain, and Behavior. MIT Press. ISBN: 9780262045353.

[5] Richard Evans, Jose Hernandez-Orallo, Johannes Welbl, Pushmeet Kohli, Marek Sergot (2020). Making sense of sensory input. arXiv:1910.02227v2 [cs.AI].