A Hybrid Active Inference Agent in a 3D Physical World: Implementation, Physics, Inference, and Control

Alexander D. Shaw
Computational Psychiatry & Neuropharmacological Systems (CPNS) Lab
Department of Psychology, Faculty of Health & Life Sciences
University of Exeter, UK
https://cpnslab.com

April 10, 2026

1 Overview

This document introduces a modular active-inference-inspired robot built in a simple 3D PyBullet environment. The broader motivation is to explore what active inference can offer for embodied intelligence: rather than treating perception, learning, and action as separate problems, the framework brings them together within a single loop in which an agent continuously updates its beliefs about the world and selects actions that make its future observations more predictable, informative, and goal-consistent.

The robot provides a concrete testbed for these ideas. It must operate in a physically realistic room-like environment, move under nontrivial dynamics, sense the world only partially, discover the location of a target object, and then return it to a home location. To do this, it combines simple embodied physics, uncertain perception, an internal world model, online belief updating, and prospective action evaluation. The result is a practical demonstration of how active-inference principles can be used to drive adaptive behaviour in an agent that must both explore and act purposefully in a continuous 3D world.

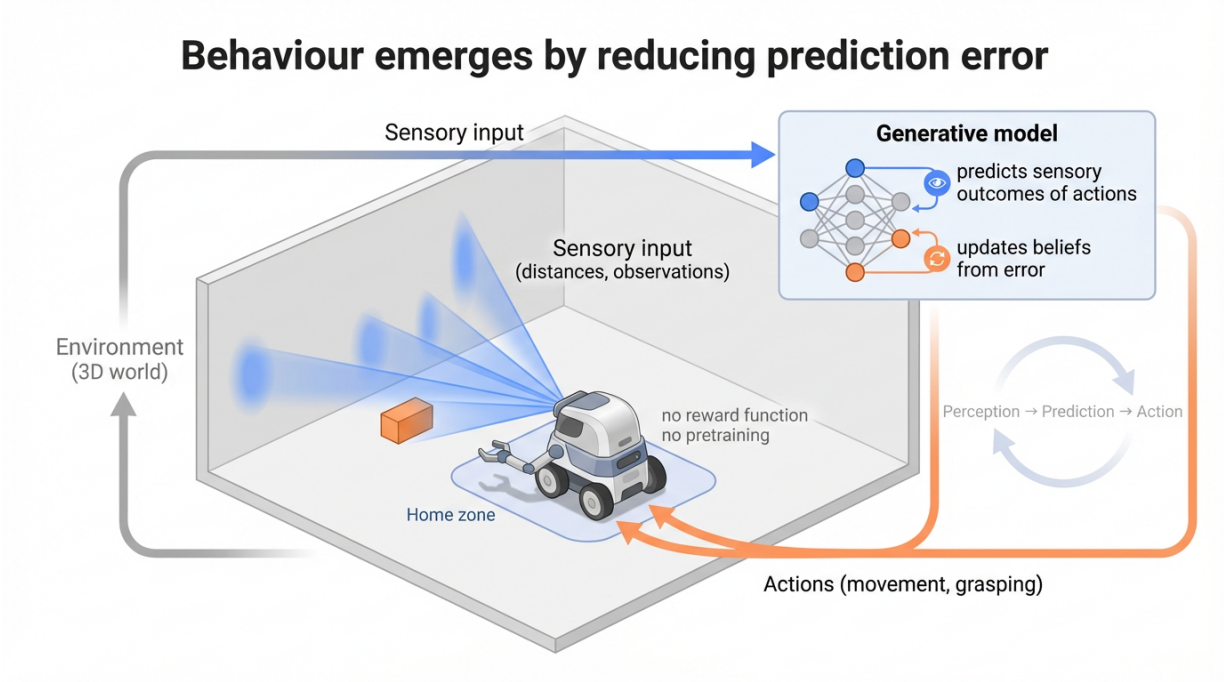

Figure 1: Conceptual overview of the collect-and-return robot. Sensory information from the 3D environment updates an internal generative model, which is then used to evaluate actions that reduce prediction error, resolve uncertainty, and support goal-directed behaviour. The agent receives no explicit reward function and no pretraining; behaviour emerges from the closed perception–prediction–action loop.

Although the system is inspired by active inference, it is best described as a *hybrid active-inference-style agent* rather than a full canonical active inference implementation. It does not perform full variational message passing over a deep generative model of hidden states and observations. Instead, it combines:

1. a belief state over pose, velocity, occupancy, and uncertainty,
2. a learned internal dynamics model,
3. short-horizon policy rollouts, and
4. a multi-term objective combining goal seeking, safety, epistemic value, efficiency, and memory/novelty.

That makes it a useful and intuitive bridge between classical active inference ideas and a practical embodied control architecture.

2 Task and Environment

The task is simple: the robot starts near a home location, navigates to an object elsewhere in the room, grasps it using a stylised arm and gripper routine, then returns the object to the home zone.

The world is implemented in PyBullet but conceptually has a planar task geometry embedded in a

3D scene. The robot moves in the horizontal plane, while the 3D scene provides walls, obstacles, furniture-like blocks, camera motion, and physically grounded collision tests. The most important configuration values are:

$$\begin{aligned} \text{room width} &= 6.0\,\text{m}, \\ \text{room height} &= 6.0\,\text{m}, \\ \Delta t &= \frac{1}{30}\,\text{s}, \\ \text{robot radius} &= 0.20\,\text{m}, \\ \text{goal radius} &= 0.40\,\text{m}, \\ \text{ray length} &= 4.0\,\text{m}, \\ \text{number of rays} &= 17, \\ \text{field of view} &= 90^\circ, \\ \text{rollout horizon} &= 5 \text{ steps}. \end{aligned}$$

The home position is initialised near

$$\boldsymbol{x}_{\text{home}} = (0.8,\, 0.8),$$

and the object is placed near

$$\boldsymbol{x}_{\text{obj}} = (4.8,\, 4.8).$$

The room contains enclosing walls and several interior obstacles. In the code these are created as static collision boxes. So although the robot is moved by a lightweight kinematic update rule, its path is constrained by collision geometry embedded in a real physics engine.

3 Robot State, Action, and Embodiment

The internal robot state used for both the environment and the learned dynamics model is

$$\boldsymbol{s}_t = \begin{bmatrix} x_t & y_t & \theta_t & v_t & \omega_t \end{bmatrix}^\top,$$

where x_t, y_t are planar position, θ_t is heading, v_t is forward velocity, and ω_t is angular velocity.

At each time step the controller chooses a 2D action

$$\boldsymbol{u}_t = \begin{bmatrix} v_t^{\text{cmd}} & \omega_t^{\text{cmd}} \end{bmatrix}^\top,$$

where the commanded forward drive is clipped to $[-1, 1]$ and the commanded turn rate is clipped to approximately $[-1.2, 1.2]$.

The robot is visualised as a small wheeled body with a front-mounted arm and two gripper fingers. These visual elements are implemented as separate PyBullet bodies that are manually repositioned each time step to match the latent planar state. This means the embodiment is visually 3D and physically grounded in collision space, but the control state itself is intentionally low-dimensional and interpretable.

4 Plant Physics and Motion Model

4.1 Environment-side plant dynamics

The actual motion update used by the environment is a first-order velocity model with inertia. Let the plant parameters be

$$(a_v, b_v, a_\omega, b_\omega, d_{\text{turn}}) = (0.86, 1.10, 0.82, 1.45, 0.16).$$

Then the environment updates velocity according to

$$\text{drag}(\omega_t^{\text{cmd}}) = \max\Big(0.2,\, 1 - d_{\text{turn}}\,|\omega_t^{\text{cmd}}|\Big)\,, \tag{1}$$

$$v_{t+1} = a_v v_t + b_v v_t^{\text{cmd}}\,\text{drag}(\omega_t^{\text{cmd}}), \tag{2}$$

$$\omega_{t+1} = a_\omega \omega_t + b_\omega \omega_t^{\text{cmd}}. \tag{3}$$

Position and heading are then advanced as

$$x_{t+1} = x_t + v_{t+1}\cos\theta_t\,\Delta t, \tag{4}$$

$$y_{t+1} = y_t + v_{t+1}\sin\theta_t\,\Delta t, \tag{5}$$

$$\theta_{t+1} = \text{wrap}(\theta_t + \omega_{t+1}\,\Delta t)\,. \tag{6}$$

This gives the robot momentum and turning inertia. The turn-drag term makes forward motion weaker during strong turns, which produces a more realistic coupling between steering and translational progress than a perfectly holonomic model.

4.2 Collision handling

Before the update is accepted, the environment performs a collision check by tracing a short ray from the current robot base position to the proposed next base position. If the ray hits an obstacle before the full movement is completed, the step is treated as blocked. In that case,

$$x_{t+1} = x_t, \tag{7}$$

$$y_{t+1} = y_t, \tag{8}$$

and the velocities are damped and partly reflected:

$$v_{t+1} \leftarrow -0.25\,v_{t+1}, \tag{9}$$

$$\omega_{t+1} \leftarrow 0.75\,\omega_{t+1}. \tag{10}$$

This is not a full rigid-body contact resolution scheme for robot locomotion. It is a pragmatic hybrid: PyBullet supplies the collision geometry and contact query, while the translational update itself remains hand-crafted and easy to reason about.

5 Observation Model

The agent does not directly observe the object as a semantic entity. Instead, the main online observation used for navigation is a set of range measurements from a fan of rays cast from the robot outward in the horizontal plane. Specifically, the ray angles are uniformly spaced over a 90° field of view:

$$\alpha_i \in \left[-\frac{\phi}{2}, \frac{\phi}{2}\right], \qquad \phi = 90^\circ, \qquad i = 1, \ldots, 17.$$

The corresponding distance measurements are

$$\boldsymbol{o}_t^{\text{ray}} = [d_{t,1}, \ldots, d_{t,17}]^\top, \qquad d_{t,i} \in [0, 4.0].$$

In addition, the observation structure passed to the agent includes:

$$\begin{aligned}
\text{touch}_t &= \mathbb{I}\left(\min_i d_{t,i} < 0.24\right), \\
\text{goal distance}_t &= \|\boldsymbol{x}_t^{\text{goal}} - \boldsymbol{x}_t^{\text{robot}}\|_2, \\
\text{goal bearing}_t &= \text{wrap}(\arctan 2(y_g - y_t, x_g - x_t) - \theta_t), \\
\text{velocity obs}_t &= [v_t, \omega_t]^\top.
\end{aligned}$$

So the observation tuple can be written as

$$\boldsymbol{o}_t = \left(\boldsymbol{o}_t^{\text{ray}}, \text{touch}_t, \beta_t^{\text{goal}}, r_t^{\text{goal}}, [v_t, \omega_t]^\top\right).$$

Notice that the goal direction and goal distance are supplied explicitly by the task wrapper. In other words, the agent does not need to infer where the current target is from raw vision. This keeps the problem focused on obstacle-aware movement, state estimation, and policy selection.

6 Belief State and Map Inference

The belief state contains:

1. a mean robot state $\boldsymbol{\mu}_t$ over pose and velocity,
2. an occupancy map over the room,
3. an uncertainty map over the same grid.

Formally,

$$\mathcal{B}_t = \{\boldsymbol{\mu}_t, \mathbf{P}_t^{\text{occ}}, \mathbf{U}_t\}.$$

6.1 State belief

The belief mean is updated very simply. The environment exposes the current pose directly, so pose is effectively observed rather than inferred:

$$\mu_t^{x,y,\theta} \leftarrow [x_t, y_t, \theta_t].$$

Velocity estimates are low-pass filtered:

$$\boldsymbol{\mu}_t^{v,\omega} = 0.70\,\boldsymbol{\mu}_{t-1}^{v,\omega} + 0.30\,[v_t, \omega_t]^\top.$$

So this part is not a fully hidden-state Bayesian filter. It is a compact belief tracker using directly available state plus filtered velocity.

6.2 Occupancy map

The map is a 72×72 grid of log-odds values:

$$L_t(i,j) \in [-4, 4].$$

Occupancy probability is obtained through a logistic transform,

$$P_t^{\text{occ}}(i,j) = \sigma(L_t(i,j)) = \frac{1}{1 + e^{-L_t(i,j)}}.$$

Each ray produces two kinds of evidence:

1. cells along free space before the hit point receive negative log-odds increments,
2. the cell at the hit point receives a positive log-odds increment if the ray terminated before maximum range.

In the code this is implemented approximately as

$$L_t(i,j) \leftarrow \text{clip}(L_{t-1}(i,j) - 0.05, -4, 4) \qquad \text{for traversed free cells,} \tag{11}$$

$$L_t(i,j) \leftarrow \text{clip}(L_{t-1}(i,j) + 0.55, -4, 4) \qquad \text{for hit cells.} \tag{12}$$

This is a classic occupancy-grid idea embedded inside the agent's internal model. It gives the controller a memory of obstacles even though each individual ray scan is only local and instantaneous.

6.3 Uncertainty map

The uncertainty map starts at one everywhere and decreases where evidence has been accumulated. For each updated cell,

$$U_t(i,j) \leftarrow \max\left(0,\, U_{t-1}(i,j) - 0.03\right).$$

This makes explored regions less uncertain and unexplored regions more uncertain. That uncertainty later contributes to an epistemic term during policy evaluation.

6.4 Visit memory

The agent also maintains a decaying visit map $V_t(i,j)$ used as a memory cost. At each step the map decays,

$$V_t \leftarrow \lambda V_{t-1}, \qquad \lambda = 0.995,$$

and the current cell gets incremented. Thus already-visited locations become mildly less attractive, encouraging exploration and helping break loops.

7 Learned Internal Dynamics Model

The agent carries its own learned transition model, separate from the environment's plant. The model has the same functional form as the plant, but with slightly different initial parameters:

$$(a_v, b_v, a_\omega, b_\omega, d_{\text{turn}}) = (0.82, 1.05, 0.76, 1.35, 0.18)$$

at initialisation.

The internal one-step prediction is

$$\hat{v}_{t+1} = a_v v_t + b_v v_t^{\text{cmd}} \max\left(0.2, 1 - d_{\text{turn}}|\omega_t^{\text{cmd}}|\right), \tag{13}$$

$$\hat{\omega}_{t+1} = a_\omega \omega_t + b_\omega \omega_t^{\text{cmd}}, \tag{14}$$

$$\hat{x}_{t+1} = x_t + \hat{v}_{t+1} \cos\theta_t \, \Delta t, \tag{15}$$

$$\hat{y}_{t+1} = y_t + \hat{v}_{t+1} \sin\theta_t \, \Delta t, \tag{16}$$

$$\hat{\theta}_{t+1} = \text{wrap}(\theta_t + \hat{\omega}_{t+1} \Delta t). \tag{17}$$

7.1 Adaptation

After observing actual state transitions, the agent adapts the model using a small learning rule. Let the prediction errors be

$$\varepsilon_v = v_{t+1}^{\text{obs}} - \hat{v}_{t+1}, \tag{18}$$

$$\varepsilon_\omega = \omega_{t+1}^{\text{obs}} - \hat{\omega}_{t+1}. \tag{19}$$

Then the parameters are adjusted proportionally to correlations between error and the relevant regressors. Schematically,

$$a_v \leftarrow a_v + \eta \, \text{clip}(\varepsilon_v v_t), \tag{20}$$

$$b_v \leftarrow b_v + \eta \, \text{clip}(\varepsilon_v \, \text{sign}(v_t^{\text{cmd}}) \, \text{drag}), \tag{21}$$

$$a_\omega \leftarrow a_\omega + \eta \, \text{clip}(\varepsilon_\omega \omega_t), \tag{22}$$

$$b_\omega \leftarrow b_\omega + \eta \, \text{clip}(\varepsilon_\omega \, \text{sign}(\omega_t^{\text{cmd}})). \tag{23}$$

with a small learning rate $\eta = 0.025$.

The turn-drag parameter is also adjusted when both translation and turning are present. All parameters are clipped to sensible bounds, preventing pathological internal models.

7.2 Calibration phase

Before the main task begins, the system runs a calibration routine. A set of seed states and canonical actions are executed in PyBullet for short rollouts, and the resulting state transitions are used to adapt the internal model. This gives the agent a rough initial alignment between its world model and the plant before real online behaviour begins.

8 Policy Library

The agent evaluates a discrete library of motor primitives rather than solving a continuous control problem from scratch at every time step. The policy set is:

$$\begin{aligned}
\texttt{forward} &: [1.00, 0.00],\\
\texttt{slow_forward} &: [0.60, 0.00],\\
\texttt{left_arc} &: [0.90, 0.45],\\
\texttt{right_arc} &: [0.90, -0.45],\\
\texttt{gentle_left} &: [0.55, 0.75],\\
\texttt{gentle_right} &: [0.55, -0.75],\\
\texttt{pivot_left} &: [0.00, 0.95],\\
\texttt{pivot_right} &: [0.00, -0.95],\\
\texttt{reverse} &: [-0.60, 0.00].
\end{aligned}$$

Each candidate is rolled forward for $H = 5$ steps using the learned transition model:

$$\hat{\boldsymbol{s}}^{(k)}_{t+1:t+H} = f^{(H)}_\theta(\boldsymbol{\mu}_t, \boldsymbol{u}^{(k)}),$$

where k indexes the policy candidate.

9 Polyphonic Objective and Action Selection

The term *polyphonic active inference* is used here to capture an important feature of the control architecture: action selection is not driven by a single objective, but by the combination of several concurrent behavioural "voices". In an embodied agent, adaptive behaviour typically requires balancing multiple pressures at once, such as approaching a target, avoiding obstacles, reducing uncertainty, preserving coherent motion, and maintaining task-relevant internal goals. Rather than treating these as separate controllers, the present framework combines them into a single prospective evaluation of candidate actions or short policies.

This is why the term *polyphonic* is useful. As in polyphonic music, behaviour emerges from the interaction of multiple simultaneous lines rather than from one dominant note. In active-inference terms, these lines can be understood as different contributors to expected future value: some terms favour epistemic actions that improve the agent's knowledge of the world, others favour pragmatic actions that move it toward preferred outcomes, while others stabilise behaviour by discouraging collisions, indecision, or loss of task structure. For each candidate policy π_k, the agent therefore computes

$$J(\pi_k) = \sum_m w_m J_m(\pi_k),$$

with weights

$$\begin{aligned} w_{\text{goal}} &= 2.2, \\ w_{\text{safety}} &= 1.7, \\ w_{\text{epistemic}} &= 0.20, \\ w_{\text{efficiency}} &= 0.30, \\ w_{\text{memory}} &= 0.55. \end{aligned}$$

The policy with lowest total score is selected.

9.1 Goal term

Let $\hat{\boldsymbol{x}}_1$ and $\hat{\boldsymbol{x}}_H$ be the predicted first and final positions under the rollout, and let $\boldsymbol{x}_g$ be the current goal. The code defines

$$\text{startDist} = \|\hat{\boldsymbol{x}}_1 - \boldsymbol{x}_g\|_2, \tag{24}$$

$$\text{endDist} = \|\hat{\boldsymbol{x}}_H - \boldsymbol{x}_g\|_2, \tag{25}$$

$$\text{progress} = \text{startDist} - \text{endDist}. \tag{26}$$

It then computes a goal cost of the form

$$J_{\text{goal}} = \text{endDist} - 2.8\,\text{progress} + 0.18\,|\Delta\theta_g|,$$

where $|\Delta\theta_g|$ is the final heading error relative to the goal direction.

So the preferred policies are those that reduce distance to the target and end up sensibly oriented.

9.2 Safety term

For each predicted state in the rollout, the local occupancy probability around the corresponding grid cell is averaged. The safety cost is then roughly

$$J_{\text{safety}} = \frac{1}{H} \sum_{h=1}^{H} \left(5\,\bar{P}_h^2 + 1.5\,\bar{P}_h\right),$$

where $\bar{P}_h$ is the local mean occupancy probability around the predicted position.

There are also direct collision penalties. If the robot is touching something and the candidate drives forward, or if the minimum current ray distance is very small and the candidate still drives forward, extra penalties are added.

9.3 Epistemic term

The epistemic term rewards predicted trajectories that pass through uncertain cells. If $\bar{U}_h$ is mean map uncertainty around predicted step h, then the code computes an epistemic value approximately as

$$\text{EV}(\pi_k) = \frac{1}{H} \sum_{h=1}^{H} \nu_h \bar{U}_h,$$

where ν_h is larger for newly encountered cells than for repeats. The corresponding cost is negative,

$$J_{\text{epistemic}} = -0.25\,\text{EV}(\pi_k).$$

This is the most recognisably active-inference-like part: uncertain, informative trajectories can become preferable because they promise learning.

9.4 Efficiency term

The efficiency term discourages unnecessarily large actions:

$$J_{\text{eff}} = 0.12|v^{\text{cmd}}| + 0.10|\omega^{\text{cmd}}| + \text{penalties for reversing and hard pivots.}$$

Reversing costs extra, and turning in place with very high angular drive also costs extra.

9.5 Memory term

The memory term discourages revisiting already frequented places:

$$J_{\text{memory}} = \gamma\,\frac{1}{H}\sum_{h=1}^{H} V(\hat{\boldsymbol{x}}_h), \qquad \gamma = 0.45,$$

with a reduced penalty if the same cell is revisited multiple times within a single rollout. This term helps the robot escape local loops and contributes a weak novelty bias.

9.6 Connection to expected free energy

This objective is not the exact textbook expected free energy

$$G(\pi) = \underbrace{\mathbb{E}_{q(o,s|\pi)}[-\ln p(o)]}_{\text{risk / preference mismatch}} + \underbrace{\mathbb{E}_{q(o,s|\pi)}[\ln q(s|\pi) - \ln q(s|o,\pi)]}_{\text{ambiguity / epistemic value related terms}},$$

but it clearly echoes the same decomposition. The goal and safety terms act like pragmatic preference terms. The epistemic term rewards uncertainty reduction. The memory term injects novelty and anti-perseveration. So the implementation is well described as an engineered approximation to active-inference-style policy evaluation.

10 Policy Commitment and Temporal Smoothing

The chosen policy is not necessarily changed every single step. Once selected, a policy is typically committed to for four time steps unless a better alternative is substantially better or the current action becomes clearly unsafe. This gives behavioural persistence and reduces twitchy oscillations.

Formally, if the current policy remains within a small score margin of the best new policy, it is retained. Otherwise the controller switches. This is not a Bayesian temporal prior in the strict sense, but it serves a similar practical role by stabilising action sequences over time.

11 Progress Monitoring and Escape Behaviour

The controller tracks progress toward the current goal using the decrease in goal distance. If progress is too small for too many steps, or if tactile/near-contact cues indicate entrapment, the system enters an explicit escape mode.

Let

$$\Delta r_t = r_{t-1}^{\text{goal}} - r_t^{\text{goal}}.$$

If $\Delta r_t < 0.03$, the no-progress counter is incremented. Touch events add further penalties. If the no-progress counter becomes large and the minimum ray distance is small, the robot enters a scripted recovery sequence:

1. reverse for a fixed number of steps,
2. pivot toward the more open side,
3. advance out of the trap.

This is another hybrid element. Instead of expecting the free-energy objective alone to solve all local entrapment, the system includes an explicit behavioural recovery routine.

12 Grasping and Post-Pickup State Machine

Navigation is continuous and active-inference-like, but grasping is discrete. The object can be picked up only when the robot is close enough and sufficiently well oriented. Specifically, if the object lies within a pickup distance of 0.42 m and within a bearing window of $\pm 40^\circ$, the grasp routine is triggered.

The grasp controller proceeds through phases:

1. **reach**: extend the arm toward the object,
2. **close**: close the gripper,
3. **retract**: retract the arm with the object attached.

If the gripper closes sufficiently, the boolean state `has_object` becomes true. Thereafter the object is no longer treated as free in the world; instead its position is slaved to the gripper tip:

$$\boldsymbol{x}_{\text{obj},t} \approx \boldsymbol{x}_{\text{tip},t} + 0.03 \begin{bmatrix} \cos\theta_t \\ \sin\theta_t \end{bmatrix}.$$

Immediately after pickup, a short post-pickup routine executes:

1. reverse,
2. pivot toward home,
3. settle forward.

This helps the robot disengage from the pickup site before normal navigation resumes.

13 Phase Switching

The task has two main navigation phases:

1. **seek**: current goal is the object,
2. **return**: current goal is home.

The current goal is therefore

$$\boldsymbol{x}_g(t) = \begin{cases} \boldsymbol{x}_{\text{obj}}, & \text{if } \texttt{has\_object} = 0, \\ \boldsymbol{x}_{\text{home}}, & \text{if } \texttt{has\_object} = 1. \end{cases}$$

Success occurs when the robot is carrying the object and is within the home radius:

$$\|\boldsymbol{x}_t^{\text{robot}} - \boldsymbol{x}_{\text{home}}\|_2 < 0.40.$$

14 Overall Algorithm

At a high level, the loop is:

1. Get current pose and velocity from the environment.
2. Cast 17 rays and build the observation tuple.
3. Update the belief state and occupancy map.
4. If grasping, continue the grasp state machine.
5. Else if in post-pickup mode, continue the post-pickup routine.
6. Else score candidate policies using short rollouts under the learned transition model.
7. Execute the selected action in the environment.
8. If conditions allow, update the learned transition model from the observed transition.
9. If the object has been acquired, switch the goal to home.
10. Stop when the robot reaches home while carrying the object.

In compact pseudo-mathematical form:

$$\begin{aligned} \boldsymbol{o}_t &\leftarrow g(\text{PyBullet world}, \boldsymbol{s}_t), \\ \mathcal{B}_t &\leftarrow \text{Infer}(\mathcal{B}_{t-1}, \boldsymbol{o}_t), \\ \pi_t^* &\leftarrow \arg\min_{\pi \in \Pi} J(\pi; \mathcal{B}_t, \boldsymbol{x}_g), \\ \boldsymbol{u}_t &\leftarrow \text{Action}(\pi_t^*), \\ \boldsymbol{s}_{t+1} &\leftarrow \text{Plant}(\boldsymbol{s}_t, \boldsymbol{u}_t), \\ \theta_f &\leftarrow \theta_f + \text{Adapt}(\theta_f; \boldsymbol{s}_t, \boldsymbol{u}_t, \boldsymbol{s}_{t+1}), \end{aligned}$$

where θ_f denotes the learned transition-model parameters.

15 Interpretation

The agent works because it combines several useful ideas in a clean way.

First, the motion model is simple but has inertia, so actions have realistic persistence. Second, the ray-based occupancy map turns local sensing into a persistent internal spatial memory. Third, the learned transition model lets the agent imagine the short-term consequences of candidate actions. Fourth, the multi-term objective balances several competing demands: go toward the goal, avoid obstacles, gather information, avoid wasteful motion, and avoid getting stuck in the same place. Finally, explicit grasp and escape routines handle edge cases that a purely reactive objective can struggle with.

In that sense, the system sits in a very nice middle ground. It is more principled and model-based than a hand-coded rule controller, but more transparent and easier to debug than a large black-box policy network.

16 Conclusion

The agent presented here is an example of how active-inference ideas can be translated into an embodied, understandable, and practical control system. The robot is not simply following rules, nor is it merely reacting to instantaneous sensor values. It maintains a world model, predicts the consequences of candidate actions, weighs multiple behavioural pressures, and uses those predictions to guide control in a structured physical environment. At the same time, it remains transparent enough that each computational ingredient can be inspected, explained, and modified.