

AIF MuJoCo Robot

Mathematical Reference

Active Inference controller for a 3D MuJoCo robot.
Beliefs, generative model, EFE, policy, and RxInfer factor graph.

1. State-Space Model

The robot operates in 3D Cartesian space (x, y, z) . The state is the position vector $s = [s_x, s_y, s_z]$. The dynamics are additive: control u shifts the state directly.

Transition Model

Given state s and control u , the next state is:

$$s' = s + u$$

With per-dimension process noise $Q = \text{diag}(q_x, q_y, q_z)$, the stochastic transition becomes:

$$s'_i = s_i + u_i + \sigma_i, \quad \sigma_i \sim N(0, q_i)$$

where $i \in \{x, y, z\}$.

Observation Model

Observations are noisy readings of the true state:

$$y_i \sim N(s_i, r_i)$$

where r_i is the observation noise variance for axis i . The observation noise can be a scalar (same for all axes) or per-axis $[r_x, r_y, r_z]$.

Source: `src/aif/generative_model.jl`

2. Belief State

The agent maintains a Gaussian belief over its 3D position with a diagonal covariance matrix:

$$b(s) = N(\mu, \text{diag}(\sigma^2_x, \sigma^2_y, \sigma^2_z))$$

where $\mu = [\mu_x, \mu_y, \mu_z]$ is the belief mean and σ^2_i is the marginal variance for axis i .

Covariance Bounds

To prevent numerical instability:

$$1 \times 10^{-5} \leq \sigma^2_i \leq 2.0$$

All covariance updates are clamped to these bounds.

Source: `src/aif/beliefs.jl` | `COV_MIN = 1e-5`, `COV_MAX = 2.0`

3. Prediction Step

Before observing, the belief is propagated forward using the applied control (the actual scaled control, not the raw action):

Mean Prediction

$$\mu'_i = \mu_i + \text{ctrl}_i$$

Covariance Prediction

$$\sigma'^2_i = \text{clamp}(\sigma^2_i + q_i, \text{COV_MIN}, \text{COV_MAX})$$

Process noise q can be a scalar (applied equally to all axes) or a 3-vector $[q_x, q_y, q_z]$. Default: $q = 0.005$.

Note: The prediction uses the actual applied control ($\text{ctrl} = \text{clamp}(\text{action} \times \text{scale})$), not the raw action, to match the true dynamics.

Source: `src/aif/beliefs.jl` | `predict_belief!()`

4. Bayesian Update (Observation)

When an observation y arrives, the belief is updated using precision-weighted fusion (conjugate Gaussian update), independently per axis:

Precision Fusion

$$\tau_{\text{prior}_i} = 1 / \sigma^2_i$$

$$\tau_{\text{lik}_i} = 1 / r_i$$

$$\tau_{\text{post}_i} = \tau_{\text{prior}_i} + \tau_{\text{lik}_i}$$

Posterior Covariance

$$\sigma^2_{\text{post}_i} = \text{clamp}(1 / \tau_{\text{post}_i}, \text{COV_MIN}, \text{COV_MAX})$$

Posterior Mean

$$\mu_{\text{post}_i} = (\tau_{\text{prior}_i} \cdot \mu_i + \tau_{\text{lik}_i} \cdot y_i) / \tau_{\text{post}_i}$$

This is the standard Kalman filter update for the diagonal case: the posterior mean is a precision-weighted average of the prior mean and the observation.

Source: `src/aif/beliefs.jl` | `update_belief!()`

5. Expected Free Energy (EFE)

Policy selection minimizes Expected Free Energy. EFE decomposes into a pragmatic (goal-seeking) term and an epistemic (exploration) term:

$$\text{EFE}(a) = G_{\text{pragmatic}}(a) - G_{\text{epistemic}}$$

Pragmatic Term (Goal-Seeking)

Measures expected squared distance to goal after applying action a . The predicted position uses the actual control scaling:

$$\text{pred}_i = \mu_i + \text{clamp}(a_i \times \text{ctrl_scale}, -\text{ctrl_lim}, \text{ctrl_lim})$$

$$G_{\text{pragmatic}} = \gamma \cdot \sum_i w_i \cdot (\text{pred}_i - \text{goal}_i)^2$$

where γ is the pragmatic weight (default 1.2) and w_i are optional per-axis weights (default [1, 1, 1]).

Epistemic Term (Exploration)

Encourages uncertainty reduction based on belief entropy:

$$G_{\text{epistemic}} = -\beta \cdot \sum_i \log(\sigma^2_i)$$

where β is the epistemic weight (default 0.05). Covariance is clamped to [1e-8, 100] for numerical stability in the log.

Combined EFE

$$\text{EFE}(a) = \gamma \sum_i w_i (\text{pred}_i - \text{goal}_i)^2 + \beta \sum_i \log(\sigma^2_i)$$

Lower EFE = better action (closer to goal + reduces uncertainty).

Source: `src/aif/efe.jl`

6. Policy Selection

The agent selects the action that minimizes EFE over a discrete action set:

$$a^* = \operatorname{argmin}_{\{a \in A\}} \operatorname{EFE}(a)$$

Action Set

$A = 5 \times 5 \times 5 = 125$ actions. Each axis has 5 velocity levels: $\{-s, -s/2, 0, s/2, s\}$ with step_size $s = 0.08$.

$$A = \{ [dx, dy, dz] : dx, dy, dz \in \{-0.08, -0.04, 0, 0.04, 0.08\} \}$$

The finer grid (compared to a 3-action set) yields smoother trajectories and better convergence.

Source: `src/aif/policy.jl`

7. Action to Control Mapping

The raw action a is scaled and clamped to produce the MuJoCo control signal:

$$\text{ctrl}_i = \text{clamp}(a_i \times \text{scale}, -\text{ctrl_lim}, \text{ctrl_lim})$$

Default: $\text{scale} = 4.0$, $\text{ctrl_lim} = 1.2$. This prevents overshooting while allowing sufficient movement per step.

The control vector $\text{ctrl} = [\text{ctrl}_x, \text{ctrl}_y, \text{ctrl}_z]$ is added to the current position in MuJoCo: $\text{target} = \text{pos} + \text{ctrl}$.

Source: *src/aif/action.jl*

8. RxInfer Factor Graph

An alternative inference backend expresses the same state-space model as a factor graph using RxInfer.jl. Each axis (x, y, z) runs an independent 1D linear-Gaussian model via reactive message passing.

Per-Axis Factor Graph

```
x_prev ~ Normal(mean = m_prev, variance = v_prev)    [prior]
x      ~ Normal(mean = x_prev + u, variance = q)     [transition]
y      ~ Normal(mean = x, variance = r)              [observation]
```

where m_{prev} and v_{prev} are the posterior parameters from the previous timestep, u is the applied control, q is process noise variance, and r is observation noise variance.

Message Passing Schedule

At each timestep t :

- Forward message: $x_{\text{prev}} \rightarrow x$ carries $N(m_{\text{prev}} + u, v_{\text{prev}} + q)$
- Likelihood message: $y \rightarrow x$ carries $N(y, r)$
- Posterior: $q(x) = N(\mu_{\text{post}}, \sigma^2_{\text{post}})$ via precision fusion

Autoupdates (Prior Propagation)

After each inference cycle, the posterior becomes the next prior:

```
m_prev, v_prev → mean(q(x)), var(q(x))
```

This is implemented with RxInfer's `@autoupdates` macro and a single atomic RecentSubject stream to avoid synchronization issues.

Equivalence to Analytic Update

For the linear-Gaussian model, belief propagation is exact. The RxInfer posterior matches the closed-form Kalman update to machine epsilon ($\approx 2.2 \times 10^{-16}$). Both backends are interchangeable.

Source: `src/aif/rxinfer_filter.jl`

9. Utility Functions

Normalize

Normalizes a probability vector in place:

$$p_i \rightarrow p_i / \sum_j p_j \quad (\text{if } \sum > 0, \text{ else uniform})$$

Gaussian PDF

Univariate:

$$p(x \mid \mu, \sigma^2) = (1 / \sqrt{2\pi\sigma^2}) \cdot \exp(-(x-\mu)^2 / (2\sigma^2))$$

Multivariate (diagonal covariance):

$$p(x \mid \mu, \sigma^2) = \prod_i p(x_i \mid \mu_i, \sigma_i^2)$$

Gaussian Entropy

Entropy of a univariate Gaussian:

$$H = 0.5 \cdot \log(2\pi e \sigma^2)$$

For the diagonal belief state, the total entropy is the sum over axes.

Softmax

Numerically stable softmax (subtract max for overflow protection):

$$\text{softmax}(x)_i = \exp(x_i - \max(x)) / \sum_j \exp(x_j - \max(x))$$

Source: `src/utils/math.jl`