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 = 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:

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

where μ = [μ_x, μ_y, μ_z] is the belief mean and σ^2_i is the marginal variance for axis i.

Covariance Bounds

To prevent numerical instability:

$$1\times10^{-5} \le \sigma^2_i \le 2.0$$

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

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

Covariance Prediction

$$\sigma^{2\prime}_i = \mathrm{clamp}(\sigma^2_i + q_i,\ \mathrm{COV\_MIN},\ \mathrm{COV\_MAX})$$

Process noise q can be a scalar (applied equally to all axes) or a 3-vector [q_x, q_y, q_z]. Default: q = 0.002.

Note: The prediction uses the actual applied control (ctrl = clamp(action × 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

$$\tau\_prior_i = 1 / \sigma^2_i$$

$$\tau\_lik_i = 1 / r_i$$

$$\tau\_post_i = \tau\_prior_i + \tau\_lik_i$$

Posterior Covariance

$$\sigma^2\_post_i = \mathrm{clamp}(1 / \tau\_post_i,\ COV\_MIN,\ COV\_MAX)$$

Posterior Mean

$$\mu\_post_i = (\tau\_prior_i \cdot \mu_i + \tau\_lik_i \cdot y_i) / \tau\_post_i$$

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:

```
EFE(a) = G_pragmatic(a) − G_epistemic
```

Pragmatic Term (Goal-Seeking)

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

```
predᵢ = μᵢ + clamp(aᵢ × ctrl_scale, −ctrl_lim, ctrl_lim)
```

```
G_pragmatic = γ · ∑ᵢ wᵢ · (predᵢ − goalᵢ)²
```

where γ is the pragmatic weight (default 1.5) and w_i are optional per-axis weights (default [1, 1, 1]).

Epistemic Term (Exploration)

Encourages uncertainty reduction based on belief entropy:

```
G_epistemic = −β · ∑ᵢ log(σ²ᵢ)
```

where β is the epistemic weight (default 0.02). Covariance is clamped to [1e-8, 100] for numerical stability in the log.

Combined EFE

```
EFE(a) = γ ∑ wᵢ(predᵢ − goalᵢ)² + β ∑ log(σ²ᵢ)
```

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* = argmin_{a ∈ A} EFE(a)
```

Action Set

A = 7×7×7 = 343 actions. Each axis has 7 velocity levels: {−3s, −2s, −s, 0, s, 2s, 3s} with step_size s = 0.04.

```
A = { [dx, dy, dz] : dx, dy, dz ∈ {-0.12, -0.08, -0.04, 0, 0.04, 0.08, 0.12} }
```

The fine 7-level grid minimises quantization artifacts for smoother trajectories. Combined with EMA control smoothing, this produces clean, jitter-free curves.

Source: src/aif/policy.jl

7. Action to Control Mapping

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

```
ctrlᵢ = clamp(aᵢ × scale, −ctrl_lim, ctrl_lim)
```

Default: scale = 3.0, ctrl_lim = 1.2. This prevents overshooting while allowing sufficient movement per step.

EMA Smoothing

Before applying the control, an Exponential Moving Average (EMA) filter smooths consecutive control signals:

```
ctrl_smooth = α · ctrl_new + (1 − α) · ctrl_prev
```

where α is the smoothing weight (default 0.3). Lower α produces smoother trajectories (more weight on previous control). α = 1.0 disables smoothing.

The control vector ctrl = [ctrl_x, ctrl_y, ctrl_z] is added to the current position in MuJoCo: target = pos + 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_prev → x carries N(m_prev + u, v_prev + q)
- Likelihood message: y → x carries N(y, r)
- Posterior: q(x) = N(μ_post, σ^2_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 ($\approx 2.2\times10^{-16}$). Both backends are interchangeable.

Source: src/aif/rxinfer_filter.jl

9. Utility Functions

Normalize

Normalizes a probability vector in place:

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

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):

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

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