FLOWRRA — Density & Loop-Collapse Design (Mathematical Spec)

Author: Sensei / Rohit (collab) **Version:** FLOWRRA v1 (design notes) **Date:** 2025-09-05 (Asia/Kolkata)

---

0. Overview

This document captures the new design for the Density Function Estimator (speed-aware / comet-tail repulsion), the **loop-level collapse** mechanism, and additional integration notes and practical choices for FLOWRRA's EnvironmentA/EnvironmentB pipeline. The goal is to: (1) preserve smooth attraction while learning repulsion online, (2) make collapse a loop-level event that snaps the whole node-loop toward the last coherent state, and (3) be computationally practical for continuous training.

---

1. Notation & primitives

- Space: positions $x \in \mathbb{R}^d$ (usually $d = 2$ or 3).
- Time: discrete steps $t \in \mathbb{Z}_{\geq 0}$.
- Node loop (full state at time t): $L(t) = [x_1(t), x_2(t), \ldots, x_N(t)]$ where N is number of nodes.
- Velocity of object or node: $v(t) \in \mathbb{R}^d$.
- Grid: world discretized into cells indexed by g . Mapping: $g = G(x)$ and inverse mapping $x \approx G^{-1}(g)$.
- Repulsion field on grid: $R(g, t) \geq 0$.
- Base (positive) potential: $U_{pos}(x)$ (smooth, fixed prior).
- Total potential: $U(x, t) = U_{pos}(x) + \beta\, r(x, t)$, where $r(x, t)$ is continuous repulsion interpolated from grid $R(g, t)$.
- Density: $\rho(x, t) \propto \exp(-U(x, t))$.
- Coherence metric: $C(L(t))$ scalar in $[0, 1]$; higher is more coherent.
- Thresholds: coherence threshold θ_c ; event severity normalization into $[0, 1]$.

---

2. Density Function Estimator — speed-aware repulsion (math + algorithm)

2.1 Principles

- Positive attraction is broad, static or slowly-changing (a smooth prior). Keep it permissive.
- Repulsion is learned online by splatting event kernels. For moving obstacles, splat **forward along velocity** to create comet-tails.
- Maintain smoothness via local diffusion (blur) and gradual decay (forgetting).

2.2 Field update equations (continuous intuition)

Let an event at time t occur at position x_t with observed velocity v_t and severity $s_t \in [0,1]$. We project repulsion forward for K_f steps ahead (to anticipate movement), and optionally backtrace for retrocausal influence of recent path (length K_b).

Define a radial-basis kernel centered at y with scale σ :

$$ \kappa_{\sigma}(x;y) = \exp\Big(-\frac{|x-y|^2}{2\sigma^2}\Big). $$

The instantaneous additive repulsion contribution at time t is:

$$ \Delta r(x,t) = \eta\, s_t \sum_{k=0}^{K_f} \gamma_f^{k} \; \kappa_{\sigma_f}(x; x_t + k\Delta t\, v_t) \, + \, \eta\, s_t \sum_{k=1}^{K_b} \gamma_b^{k} \; \kappa_{\sigma_b}(x; x_{t-k}), $$

where:

- η = learning rate for repulsion splats (e.g., 0.05–0.3).
- $\gamma_f \in (0,1]$ = forward decay for projected splats (closer future stronger), typical 0.6–0.95.
- K_f = number of forward steps to project (small integer, e.g., 3–10).
- Δt = time-step scaling for projection (in world units per step).
- $\gamma_b \in (0,1]$ = retro/backtrace decay for past path reinforcement.
- K_b = backward trace length (optional; 0 or small like 5–20).
- σ_f, σ_b = kernel widths for forward/back kernels.

2.3 Discrete grid implementation

On the grid, expressive update per time-step for each event is:

1. Splat: for each projected center $y_k = x_t + k\Delta t\, v_t$, add to grid cell(s):

$$ R(g,t) \leftarrow R(g,t) + \eta\, s_t\, \gamma_f^{k} \, \kappa_{\sigma_f}(G^{-1}(g); y_k) $$

1. Retro splat (if used): similar with x_{t-k} centers and decay γ_b^k .
2. Decay (forget old scars):

$$ R(g,t+1) = (1 - \lambda)\, R(g,t) + \text{SplatAdds}(g)\quad\text{with }\lambda\in(0,1)\,. $$

1. Smooth (diffuse) — small convolution / blur step to keep field differentiable:

$$ R(g,t+1) \leftarrow (1 - \delta)\, R(g,t+1) + \delta\, (B * R)(g,t+1) $$

where $B * R$ denotes a small-kernel convolution (Gaussian blur) and $0 < \delta \ll 1$. 5. Clip to bounds: $R(g,t+1) \leftarrow \min(R(g,t+1), R_{max})$.

2.4 From potential to density

Interpolate repulsion at continuous point x :

$$ r(x,t) = \text{interpGrid}(R(\cdot,t), x). $$

Total potential:

$$ U(x,t) = U_{pos}(x) + \beta\, r(x,t). $$

Density used for sampling or probabilistic evaluation:

$$ \rho(x,t) = \frac{\exp(-U(x,t))}{\int \exp(-U(x,t))\, dx}. $$

Practically, you can skip exact normalization when using gradient descent on U or when comparing relative densities.

---

3. Loop-level collapse (math + algorithm)

3.1 Motivation

Collapse must preserve the **entire loop** structure. Single-node collapse fragments the loop and creates perpetual failures. We make the collapse a discrete operator acting on $L(t)$ conditioned on a loop coherence metric.

3.2 Coherence metric — examples

Choose one or a composite of these:

- **Density-consistency**: measure how well nodes sit in high-density regions: $C_\rho(L) = \frac{1}{N}\sum_{i=1}^{N}\frac{\rho(x_i)}{\rho_{max}} \in [0,1].$
- **Geometric energy**: loop energy (prefer neighbor spacing and angle consistency): $E_{loop}(L) = \sum_{i=1}^{N}\Big(w_d\|x_{i+1}-x_i\|^2 + w_\theta(1-\cos\theta_i)\Big)$, with normalized transform to give $C_E(L) \in [0,1]$.
- **Spectral coherence**: measure eigenstructure stability of adjacency/graph Laplacian over the loop.

Combine if desired: $C(L) = \alpha_1 C_\rho + \alpha_2 C_E + \alpha_3 C_{spec}$.

3.3 Collapse rule (discrete)

Let $C(L(t))$ be computed each step. If $C(L(t)) < \theta_c$ then trigger collapse.

Collapsed state choice options:

1. **Rollback to last coherent state**: maintain a short buffer $\{L(t-\tau)\}$ of previous loop states and choose the latest $L^* = L(t-\tau^*)$ with $C(L^*) \ge \theta_c$.
2. **Projection to nearest coherent manifold**: compute a constrained optimization:

$$ L^* = \arg\min_{L' \in \mathcal{S}} |L' - L(t)|^2 \quad\text{s.t. } C(L') \ge \theta_c, $$

where $\mathcal{S}$ may encode loop topology constraints (circularity/order), neighbor distances, etc. 3. **Hybrid**: rollback a bit and then project small adjustments.

Choose (1) for simplicity and robustness. Keep a short history buffer size H and a timestamped coherence label.

3.4 Algorithm (loop collapse)

1. Each step compute $C(L(t))$.
2. If $C(L(t)) \ge \theta_c$: append $L(t)$ to coherent buffer.
3. Else (collapse): set $L(t) \leftarrow L^*$ where L^* is last coherent state from buffer (or projection if no buffer). Definitely, add a repulsion scar originating from positions in the collapsed trajectory (severity scaled by failure magnitude).
4. Penalize / learn from collapse: create event with severity $s_t = 1 - C(L(t))$ and call repulsion splat update (section 2) with retro/backtrace **along the loop path** (so entire loop is discouraged from repeating the failing trajectory).

3.5 Implementation notes

- Keep buffer H small but long enough to store last good shapes (e.g., H=50–200 steps depending on dynamics).
- If buffer empty, project onto a canonical loop (e.g., circular attractor) to avoid deadlock.
- Maintain versioning/time stamps to prevent oscillatory flip-flop between nearby states.

---

4. Integrating both pieces: practical API and pseudocode

4.1 Module: `RepulsionField`

State: grid `R`, params `eta, sigma_f, gamma_f, K_f, delta, lambda, R_max`. **Methods:** `splat_event(pos, vel, severity)`, `decay_and_blur()`, `sample_potential(x)`, `save/load`.

4.2 Module: `LoopManager`

State: history buffer of loop states `buffer`, coherence function `compute_C(L)`, threshold `theta_c`. **Methods:** `step(L_current) -> L_next` which applies collapse logic and returns possibly replaced `L_next`.

4.3 Training loop sketch

```
for episode in range(N_episodes):
  reset env
  L = initial_loop()
  for t in range(T):
    # 1. sense world (obstacles, velocities)
    events = sense_events()
    for e in events:
      RepulsionField.splat_event(e.pos, e.vel, e.severity)
    RepulsionField.decay_and_blur()

    # 2. compute potentials & let nodes move (policy or gradient)
    for i in range(N):
      x = L[i]
      U = U_pos(x) + beta * RepulsionField.sample_potential(x)
      # e.g., gradient descent step: x <- x - alpha * grad U
    L = apply_dynamics(L)

    # 3. loop-level coherence + collapse
    L = LoopManager.step(L)

    # 4. when collapse triggers, create retro repulsion along loop path
    if collapse_happened:
      RepulsionField.splat_event_along_path(loop_path, severity)
```

---

5. Additional design points & heuristics

1. **Parameter table (initial suggestions)**
2. `eta`: 0.05 – 0.2
3. `sigma_f`: 0.5 – 2.0 (world units)
4. `gamma_f`: 0.6 – 0.95
5. `K_f`: 3 – 10
6. `lambda` (decay): 0.001 – 0.01 per step
7. `delta` (blur mix): 0.05 – 0.15
8. `beta` (repulsion weight): 0.5 – 2.0 (anneal upwards)
9. `H` (coherent buffer): 50 – 200
10. **Multi-node aggregation**

11. If many nodes report events simultaneously, aggregate via log-sum-exp to preserve soft consensus; or use `max` for hard shared barriers.

12. **Computational complexity**

13. Splat cost is proportional to number of projected centers (K_f) times kernel footprint on grid; keep kernels small and use sparse updates.

14. Blur is small-kernel convolution (constant small cost per step). Grid resolution trades precision vs compute.

15. **Testing & validation metrics**

16. Collapse frequency per epoch (goal: drop from near 100% to rare events).

17. Average coherence $\bar{C}$ over time.
18. Collision/entropy events per episode.

19. Stability of loop energy E_{loop} .

20. **Retrocausal shaping vs. obstacle projection**

21. Both use the same splat mechanics but with different directionality: backward splats for retrocausal learning, forward splats for obstacle anticipation.

22. **Hard mask fallback**

23. If severity > emergency threshold (e.g., repeated collapse in a small region), add a temporary hard mask (zero probability region) for a short duration to force safe retraining.
