The Neural-Klotski System

---

1. Overview

The Neural-Klotski System is a computational framework that maps neural computation onto a spatial-mechanical substrate where computation manifests as physical movement through configuration space.

The system consists of blocks (representing neurons) that slide along two orthogonal axes: an activation axis encoding membrane potential, and a lag axis encoding temporal phase. Blocks are connected by colored wires (representing synapses) that transmit discrete signals when source blocks cross individual threshold positions. Wire color is determined by the source block's identity: red wires transmit excitatory signals that push target blocks toward firing, while blue wires transmit inhibitory signals that push targets away from firing. The fundamental insight is that every possible arrangement of blocks constitutes a distinct computational state, and the complete graph of all possible configurations forms a pre-existing landscape containing all potential computations. Thinking, in this framework, is navigation through this configuration space.

This document provides a complete technical specification of the system: its components, dynamics, plasticity mechanisms, and input/output interfaces. The specification is self-contained and sufficient for implementation in either physical or simulated form.

---

2. System Architecture

2.1 Spatial Dimensions

The system operates in a two-dimensional coordinate space with distinct functional roles for each axis.

Activation Axis (Horizontal): The activation axis represents the membrane potential or excitation level of a neuron. Position along this axis is continuous and bounded. The origin (position 0) represents the resting state. Positive positions represent depolarization (movement toward firing), while negative positions represent hyperpolarization (inhibited or refractory states). The axis has defined bounds: blocks cannot move beyond a maximum rightward position or below a minimum leftward position, representing physical voltage limits.

Lag Axis (Vertical): The lag axis represents temporal phase or timing offset within the network. Each block occupies a fixed position along this axis (though this position is modifiable through learning). The lag position determines when a block effectively receives and processes signals relative to other blocks in the network. Blocks at different lag positions experience signals at different effective times, creating inherent propagation delays and enabling temporal pattern processing. Higher lag values correspond to later processing times.

Coordinate System: A block's complete spatial state is defined by its position on the activation axis (which changes dynamically during computation) and its position on the lag axis (which remains fixed during computation but can be adjusted during learning). The configuration of all blocks across both dimensions defines the complete system state.

2.2 Components

The system consists of four fundamental component types:

Blocks: Discrete units that slide along the activation axis while maintaining fixed positions on the lag axis. Each block has intrinsic properties including a color identity (red, blue, or yellow), a threshold value, and physical dynamics (position, velocity, refractory state). Blocks represent individual neurons.

Wires: Connections between blocks that transmit discrete signals. Each wire connects a source block to a target block and has a strength parameter determining signal amplitude. Wire color is determined entirely by the source block's identity: all wires emanating from a red block are red (excitatory), all wires from a blue block are blue (inhibitory), and all connections from a yellow block are yellow (electrical coupling). Wires represent synapses.

Dyes: Diffusible substances that spread spatially through the system and temporarily modulate wire effectiveness. Dyes have color identities (red, blue, yellow) and selectively enhance wires matching their color. Dye concentration varies spatially and decays exponentially over time. Dyes represent neuromodulatory signals like dopamine or serotonin and serve as learning signals that gate plasticity.

Network Topology: The pattern of wire connections between blocks defines the computational architecture. This topology is specified at initialization and determines which blocks can influence which other blocks. The topology can be sparse (few connections) or dense (many connections), local (nearby blocks connected) or global (distant blocks connected), and may include feedback loops (recurrent connections).

2.3 Configuration Space (Theoretical Note)

The complete system state at any moment is defined by the activation positions of all blocks. The set of all possible system states forms a configuration space, and computation can be viewed as traversing paths through this space. Each possible arrangement of blocks represents a distinct computational state. This theoretical framework—where the space of all possible

thoughts pre-exists and computation is navigation through it—provides conceptual insight but is not required for implementation. For practical purposes, the system state is simply the vector of all block positions and their associated properties at a given time.

---

3. Block Specification

3.1 Physical Properties

Each block in the system is characterized by the following properties:

Position on Activation Axis: A continuous scalar value representing the block's current location along the horizontal axis. This value changes dynamically during system operation. Position is measured relative to the origin (resting state at position 0). Positive values indicate depolarization, negative values indicate hyperpolarization or refractory states.

Position on Lag Axis: A continuous scalar value representing the block's fixed vertical location. This value remains constant during normal operation but can be modified during learning phases. The lag position determines the temporal offset at which the block processes incoming signals.

Velocity: The rate of change of position along the activation axis. Velocity is used in the continuous dynamics equations governing block movement. Velocity is affected by forces from incoming signals, spring forces pulling toward rest, and damping.

Color Identity: Each block has one of three color identities that determines its functional role:

- **Red blocks:** Excitatory neurons. All outgoing wires are red and transmit excitatory signals.
- **Blue blocks:** Inhibitory neurons. All outgoing wires are blue and transmit inhibitory signals.
- **Yellow blocks:** Electrical coupling neurons. All connections are yellow and represent gap junctions (bidirectional, continuous coupling).

The color identity is assigned at initialization and does not change.

Threshold Value: A scalar value representing the position on the activation axis at which the block fires. When a block's position crosses this threshold from left to right, a firing event is triggered. Each block has its own individual threshold value, allowing for heterogeneous excitability across the network.

Refractory Timer: A scalar value tracking the remaining duration of the refractory period. When a block fires, this timer is set to a positive value and counts down. While the timer is positive, the block is in a refractory state.

3.2 States and Transitions

A block exists in one of four primary states at any given time:

Resting State: The block is at or near the origin (position ≈ 0) on the activation axis with minimal velocity. The block is responsive to incoming signals and can move toward threshold if stimulated. This is the default state to which blocks return after activity.

Depolarizing State: The block's position is between the origin and its threshold value, moving rightward due to excitatory input or leftward due to inhibitory input. The block has non-zero velocity and is actively responding to forces from wires and internal spring forces. The block has not yet fired and can fire if it crosses threshold.

Firing State: The block has crossed its threshold position. This triggers a discrete firing event: all outgoing wires become active and transmit signals to their target blocks. Immediately upon firing, the block transitions to the refractory state.

Refractory State: Following a firing event, the block receives a strong negative velocity kick that propels it leftward into negative positions on the activation axis. The refractory timer is active. During this period, the block is positioned far from threshold, making firing highly unlikely. Incoming signals still apply forces and can influence the block's position, but the block must return close to origin and the refractory timer must expire before firing is possible again. The block gradually returns to the origin under spring forces. Once the refractory timer expires and position returns near the origin, the block transitions back to resting state.

State Transitions:

- Resting → Depolarizing: Incoming signal provides force, moving block away from origin
- Depolarizing → Firing: Position crosses threshold value from below
- Firing → Refractory: Immediate transition upon threshold crossing
- Refractory → Resting: Refractory timer expires and position returns to near-origin
- Depolarizing → Resting: If forces diminish, block returns to origin without firing

3.3 Block Dynamics Summary

Block behavior is governed by continuous dynamics (position and velocity evolving under forces) punctuated by discrete events (threshold crossing triggers firing and refractory snap). The color identity determines output signal type, the threshold determines excitability, and the lag position determines temporal processing characteristics. All blocks follow the same physical laws but differ in their parameters, creating computational diversity.

3.4 Dye Specification

3.4.1 Dye Properties

Color Identity: Each dye instance has one of three color identities:

- **Red dye:** Enhances red (excitatory) wires only
- **Blue dye:** Enhances blue (inhibitory) wires only
- **Yellow dye:** Enhances yellow (electrical coupling) connections only

Spatial Distribution: Dye exists as a concentration field over the spatial extent of the system. At any position (x, y) in the system, there is a dye concentration value C(x, y, t) for each color, representing how much of that dye color is present at that location and time.

Concentration Values: Dye concentration is a continuous non-negative scalar. Typical values range from 0 (no dye present) to 1.0 (maximum concentration). Concentrations above 1.0 are possible but typically clamped to prevent excessive enhancement.

Temporal Evolution: Dye concentration at any location changes over time according to diffusion (spatial spreading) and decay (temporal reduction).

3.4.2 Dye Dynamics

Diffusion: Dye spreads spatially from regions of high concentration to regions of low concentration. The diffusion follows:

$\partial C/\partial t = D \times \nabla^2 C$

Where D is the diffusion coefficient and $\nabla^2 C$ is the spatial Laplacian (second spatial derivative). In discrete simulation, this can be approximated by averaging concentration with neighboring spatial positions.

Practical effect: Dye released at a point spreads outward over time, creating a concentration gradient. The diffusion rate determines how quickly dye spreads through the system.

Decay: Dye concentration decreases exponentially over time according to:

$C(t) = C(0) \times e^{(-t/\tau)}$

Where τ is the decay time constant. Typical values: τ = 100-1000 timesteps, meaning dye persists for seconds to minutes in simulation time before decaying to negligible levels.

Injection: Dye can be injected into the system at specific locations and times. An injection event adds a specified concentration amount to the dye field at the injection location:

C(x_inject, y_inject, t_inject) += injection_amount

Multiple injections can occur, and their effects sum spatially and temporally according to diffusion and decay dynamics.

3.4.3 Wire Enhancement Mechanism

Color-Selective Enhancement: Dye only affects wires matching its color. Red dye has no effect on blue or yellow wires, blue dye has no effect on red or yellow wires, and yellow dye has no effect on red or blue wires.

Local Concentration Lookup: Each wire samples the local dye concentration at its spatial location. For a wire at position (x_wire, y_wire), the relevant dye concentration is C_color(x_wire, y_wire, t) where color matches the wire's color.

Effective Strength Calculation: The effective strength of a wire in the presence of dye is:

effective_strength = base_strength × (1 + enhancement_factor × C_local)

Where:

- base_strength is the wire's intrinsic strength parameter
- enhancement_factor determines how strongly dye affects wires (typically 1.0 to 3.0)
- C_local is the local dye concentration of matching color

Example: A red wire with base_strength = 2.0 in a region with red dye concentration 0.5 and enhancement_factor = 2.0: effective_strength = 2.0 × (1 + 2.0 × 0.5) = 2.0 × 2.0 = 4.0

The wire is twice as strong as its baseline.

No Effect on Mismatched Colors: If a wire's color doesn't match the dye, C_local = 0 for that wire regardless of dye presence, so effective_strength = base_strength.

---

4. Wire Specification

4.1 Properties

Each wire in the system is characterized by the following properties:

Source Block: The block from which the wire originates. The source block's firing events trigger signal transmission through this wire.

Target Block: The block to which the wire connects. This block receives signals transmitted through the wire and experiences forces as a result.

Color: The wire's color is determined entirely by the source block's color identity. A wire emanating from a red block is a red wire, a wire from a blue block is a blue wire, and a connection from a yellow block is a yellow wire. Wire color determines the type of signal transmitted and its effect on the target.

Strength Parameter: A scalar value determining the amplitude of the signal transmitted through the wire. Larger strength values produce stronger effects on the target block. Strength can be positive (for red and yellow wires) or have its sign determined by wire color (blue wires inherently produce inhibitory effects). This parameter is subject to modification through plasticity mechanisms.

Signal Type: Determined by wire color:

- **Red wires (Excitatory):** Transmit signals that apply rightward force to the target block, pushing it toward its threshold.
- **Blue wires (Inhibitory):** Transmit signals that apply leftward force to the target block, pushing it away from its threshold.
- **Yellow wires (Electrical):** Represent gap junctions with bidirectional, continuous coupling between blocks. Unlike red and blue wires which transmit discrete signals upon firing, yellow connections create continuous force proportional to the position difference between connected blocks.

4.2 Signal Transmission

Firing Condition: For red and blue wires, signal transmission occurs when the source block enters the firing state (crosses its threshold from below). The wire becomes active and transmits a discrete signal to its target.

Signal Propagation: When a wire becomes active, a signal is generated and propagates to the target block. Signals travel at finite speed along the lag axis. The propagation delay depends on the distance between source and target lag positions.

Propagation Delay:

lag_distance = |λ_target - λ_source|

propagation_delay = lag_distance / v_signal

Where v_signal is the signal propagation speed (typically 50-200 lag units per timestep).

Effect on Target: Upon signal arrival at the target block, a force is applied:

- **Red wire signal:** Applies a rightward force proportional to the wire's strength parameter. The force magnitude is: F = +strength
- **Blue wire signal:** Applies a leftward force proportional to the wire's strength parameter. The force magnitude is: F = -strength
- **Yellow wire (continuous):** Applies a force proportional to the position difference between source and target blocks: F = strength × (source_position - target_position). This creates a coupling that tends to synchronize the positions of connected blocks.

Timing and Lag: The effective timing of signal arrival depends on the lag positions of the source and target blocks. A signal from a source at lag position L_source to a target at lag position L_target experiences a delay related to the difference (L_target - L_source). Blocks at higher lag positions effectively receive signals later than blocks at lower lag positions, even when signals are transmitted simultaneously. This creates temporal structure in signal processing.

4.3 Wire Activation Rules

Red and Blue Wires:

- Inactive by default
- Become active when source block fires
- Signal queued for delivery after propagation delay
- Return to inactive state after signal delivered
- Can only transmit signals when source crosses threshold from below

Yellow Wires:

- Always active (continuous coupling)
- No firing event required
- Force continuously present based on position difference
- Bidirectional: both blocks experience forces toward each other
- No propagation delay (instantaneous coupling)

4.4 Multiple Wire Effects

A single target block may receive signals from multiple wires simultaneously. When this occurs, the forces from all active wires are summed. The net force on the target block is the vector sum of all individual wire forces, allowing for:

- **Summation:** Multiple excitatory signals combine to produce stronger depolarization
- **Cancellation:** Excitatory and inhibitory signals can partially or fully cancel each other
- **Competition:** Different sources competing to influence the target's state

---

5. System Dynamics

5.1 Block Movement

Block position on the activation axis evolves according to continuous dynamics governed by forces, damping, and physical constraints.

Equation of Motion: The position of a block changes according to its velocity, and velocity changes according to applied forces with damping. For a block i with position x_i and velocity v_i:

Position update: $x_i(t + dt) = x_i(t) + v_i(t) \times dt$

Velocity update: $v_i(t + dt) = v_i(t) \times \text{damping\_coefficient} + F_{total} \times dt$

Where damping_coefficient is a value between 0 and 1 (typically 0.8-0.95) that causes velocity to decay over time, creating overdamped motion.

Forces: The total force F_{total} acting on a block is the sum of multiple components:

Spring Force: A restoring force that pulls the block toward the origin (resting position). This force is proportional to the block's distance from origin: $F_{spring} = -k \times x_i$

Where k is the spring constant. This ensures blocks naturally return to rest when not stimulated.

Wire Forces: Forces from incoming signals via wires. When a wire targeting block i is active: $F_{wire} = \text{strength} \times \text{sign}$

Where sign is +1 for red (excitatory) wires, -1 for blue (inhibitory) wires. For yellow wires (gap junctions), the force is proportional to position difference between source and target.

Total Force Calculation: $F_{total} = F_{spring} + \Sigma(F_{wire}$ for all active incoming wires$)$

The sum includes all wires targeting this block that are currently active.

Damping: The damping coefficient ensures that blocks do not oscillate indefinitely and that motion is overdamped (heavily damped, like moving through viscous fluid). This creates smooth, controlled movement rather than oscillatory behavior.

Boundary Conditions: The activation axis is effectively unbounded. While extremely large positive or negative positions are not feasibly reachable under normal dynamics due to the restoring spring force, there are no hard boundaries that constrain block movement.

5.2 Threshold Crossing

Detection: At each time step, the system checks whether any block has crossed its threshold. Crossing is detected when:

- Previous position: $x_i(t) < \text{threshold}_i$
- Current position: $x_i(t + dt) \geq \text{threshold}_i$
- Block is not currently in refractory state (refractory timer > 0)
- **Critical:** Block must cross from below (position was below threshold previously)

Firing Trigger: When threshold crossing from below is detected, a firing event is immediately triggered:

1. The block enters the firing state
2. All outgoing wires from this block become active
3. Signals are generated and queued for transmission to target blocks after propagation delays
4. The firing state persists for one timestep

Immediate Refractory Transition: Following the firing timestep, the block immediately transitions to refractory state. This transition involves a spring-like snap back:

1. A strong negative velocity is imparted to the block (refractory kick, typically velocity = -15 to -25)
2. This causes the block to move rapidly leftward into negative positions
3. Refractory timer is set to a positive value (refractory duration, typically 20-50 timesteps)
4. All outgoing wires become inactive

The refractory kick is a continuous, physics-based event rather than a discontinuous position jump. The block naturally moves into the refractory region due to the applied negative velocity.

5.3 Refractory Dynamics

Refractory Motion: Following the refractory kick, the block moves leftward (into negative positions) with high velocity. As it moves, the standard dynamics continue to apply:

- Spring force pulls it back toward origin
- Damping reduces velocity
- Incoming wire forces can still influence position and velocity

The block trajectory is typically: rapid leftward motion → deceleration → reversal → gradual return toward origin.

Refractory Timer: The timer decrements by 1 each timestep. While the timer is greater than zero, threshold crossing detection is disabled for this block. Even if the block's position crosses the threshold value during refractory (due to strong excitatory input), no firing event occurs.

Strong Input During Refractory: If the block receives strong excitatory input during refractory, incoming wire forces can:

- Slow or reverse the leftward motion
- Accelerate return to positive positions
- In extreme cases, push the block back toward threshold even before timer expires

However, the block cannot fire until the refractory timer reaches zero.

Exit Condition: The block exits the refractory state and can fire again when the refractory timer reaches zero. At this point, if the block's position is above threshold (due to strong sustained input during refractory), it will immediately fire again. This enables burst firing patterns.

Burst Firing: Strong sustained excitatory input can create burst patterns:

1. Block fires → refractory kick sends it leftward
2. Strong input continues during refractory
3. Input forces push block back rightward during refractory
4. Timer expires while block is near or above threshold
5. Block fires again immediately (must cross from below)
6. Cycle repeats, creating rapid burst of firing events

5.4 Signal Propagation and Temporal Sequencing

5.4.1 Signal Generation

When a block fires, signals are generated on all its outgoing wires. Each signal carries the wire's strength parameter and type (excitatory/inhibitory).

5.4.2 Signal Propagation Delay

Signals transmitted along wires are not instantaneous. They propagate at finite speed along the lag axis.

Propagation speed: Signals travel at a fixed speed v_signal (typically 50-200 lag units per timestep). The time required for a signal to travel from source to target depends on the distance between their lag positions.

Delay calculation:

For wire j connecting source block s_j to target block t_j:

$$\text{lag\_distance} = |\lambda_{t_j} - \lambda_{s_j}|$$

$$\text{propagation\_delay} = \text{lag\_distance} / v_{signal}$$

Signal transmission and arrival:

When source block s_j fires at time t_fire:

$$t_{arrival} = t_{fire} + \text{propagation\_delay}$$

Signal queued for delivery at t_arrival

Force applied to target block at t_arrival, not immediately

Implementation: The system maintains a signal queue containing pending signals with their arrival times. Each timestep, signals whose arrival time matches the current time are processed and their forces applied to target blocks.

Effect on computation: The lag axis creates genuine temporal structure. Blocks at higher lag positions receive signals later than blocks at lower lag positions, even when source blocks fire simultaneously. This enables:

- Temporal pattern detection (sequences of firing events)
- Integration windows (later blocks collect inputs from earlier blocks)
- Propagation cascades (activity flows through network over time)
- Phase relationships (synchronization and relative timing)

The finite propagation speed is a fundamental feature that distinguishes blocks at different lag positions temporally, not just spatially.

5.4.3 Coincidence Detection

When multiple signals arrive at a target block within a short time window, their forces sum. If the combined force is sufficient, the target crosses threshold and fires. This implements coincidence detection: the target responds preferentially to synchronized inputs.

5.4.4 Temporal Integration

Blocks at higher lag positions receive signals from multiple sources with varying delays, allowing them to integrate information over time. This creates a natural mechanism for temporal pooling and sequence recognition.

5.4.5 Dye Modulation of Signal Strength

When a wire transmits a signal, the force applied to the target block is determined by the wire's effective strength, which includes dye enhancement:

F_wire = effective_strength × sign

Where effective_strength incorporates local dye concentration as specified in Section 3.4.3.

Effect on Computation: Regions with matching dye color experience amplified signal transmission. This creates temporary functional changes in network behavior:

- Red dye regions: excitatory pathways strengthened, easier to drive blocks to threshold

- Blue dye regions: inhibitory pathways strengthened, easier to suppress activity
- Yellow dye regions: electrical coupling strengthened, increased synchronization

Spatial Specificity: Dye enhancement is spatially local. Wires in high-concentration regions are strongly enhanced, wires in low-concentration regions are weakly enhanced, and wires outside the dye cloud are unaffected.

Temporal Dynamics: As dye diffuses and decays, the pattern of enhancement changes. Initially concentrated enhancement spreads and weakens over time, creating a dynamic landscape of modulation.

5.5 Yellow Wire (Gap Junction) Dynamics

Yellow wires operate continuously rather than through discrete firing events. At every timestep, for each yellow wire connecting blocks i and j:

Force on block i: F_i = strength × (x_j - x_i) Force on block j: F_j = strength × (x_i - x_j)

This creates bidirectional coupling where both blocks experience forces pulling them toward each other's positions. Strong yellow connections cause blocks to move in synchrony, while weaker connections create loose coupling.

Computational Role of Yellow Wires:

Yellow wires enable several computational capabilities:

Synchronization: Strongly coupled blocks tend to move together, creating synchronized firing patterns. This is useful for binding related signals or creating oscillatory behavior.

Fast Communication: Unlike red and blue wires which require threshold crossing to transmit signals, yellow wires transmit continuous positional information instantly. This enables rapid coordination without discrete firing events.

Noise Reduction: Coupling multiple blocks averages out random fluctuations, creating more stable and reliable signals.

Oscillation Generation: Networks of yellow-connected blocks can generate spontaneous oscillations, creating intrinsic rhythms useful for timing and pattern generation.

Direct Influence: Yellow connections allow blocks to directly influence each other's state continuously, rather than through discrete pulse transmission. This creates different computational dynamics compared to synaptic transmission.

In biological terms, gap junctions enable rapid synchronization of neural populations, generation of brain rhythms, and direct electrical communication that complements chemical synaptic transmission.

5.6 Complete Update Cycle

A single timestep of system evolution proceeds as follows:

1. For each block, calculate total force from spring and active incoming wires
2. Update velocity using force and damping
3. Update position using velocity
4. Check for threshold crossings; trigger firing events if detected
5. For blocks that just fired, apply refractory kick (impart negative velocity) and set refractory timer
6. For blocks in refractory, decrement timer
7. Update wire activation states based on source block firing states
8. Advance time by dt

This cycle repeats continuously, driving the system's computational dynamics.

---

6. Plasticity Mechanisms

6.1 Wire Strength Adaptation

Wire strength parameters can change over time through plasticity mechanisms. Critically, long-term structural changes are gated by dye presence, implementing a two-stage learning process.

6.1.1 Hebbian Learning (Dye-Gated)

Wire strength increases when the source and target blocks are co-active AND the wire is in a region with matching dye present. The learning rule is:

Δstrength = learning_rate × dye_factor × source_activity × target_activity

Where:

- learning_rate is the baseline plasticity rate
- dye_factor = (1 + dye_enhancement × C_local) amplifies learning in dye-rich regions
- source_activity indicates whether source block fired recently
- target_activity indicates whether target block fired recently

Key insight: Hebbian learning occurs continuously, but it is dramatically amplified in regions marked by dye. Wires that are temporarily enhanced by dye AND show correlated activity undergo strong permanent strengthening.

Spike-Timing Dependent Plasticity (STDP): A refinement where temporal order matters:

If source fires BEFORE target (within temporal window): Δstrength = +learning_rate × dye_factor (potentiation)

If source fires AFTER target (within temporal window): Δstrength = -learning_rate × dye_factor (depression)

The dye_factor amplifies both potentiation and depression, accelerating learning of temporal relationships in marked regions.

6.1.2 Dye-Independent Mechanisms

Some plasticity mechanisms operate independently of dye:

Fatigue and Tearing: Repeated use causes temporary weakening:

damage_accumulation += use_rate × |signal_strength| effective_strength = base_strength × (1 - damage / max_damage)

This occurs regardless of dye presence and implements short-term synaptic depression.

Recovery and Repair: Unused wires gradually recover:

damage = max(0, damage - repair_rate × dt)

6.1.3 Long-Term Structural Changes

Permanent changes to base_strength accumulate through repeated dye-gated Hebbian learning:

base_strength(t + Δt) = base_strength(t) + Σ(Δstrength from Hebbian events)

Over many learning trials, temporary dye enhancement → repeated Hebbian strengthening → permanent structural change.

Decay of Unused Connections: Wires that are rarely active and not in dye-marked regions slowly weaken:

If wire unused for extended period: base_strength = base_strength × (1 - decay_rate)

Strength Bounds: minimum_strength ≤ base_strength ≤ maximum_strength

Typical bounds: 0.1 to 10.0

6.2 Threshold Adaptation

Block threshold values can adjust based on activity history, implementing homeostatic plasticity.

Activity-Dependent Adjustment: If a block fires very frequently, its threshold increases (becomes less excitable):

If firing_rate > target_firing_rate: Δthreshold = +adaptation_rate × (firing_rate - target_firing_rate)

If a block rarely fires, its threshold decreases (becomes more excitable):

If firing_rate < target_firing_rate: Δthreshold = -adaptation_rate × (target_firing_rate - firing_rate)

Where:

- firing_rate is measured over a sliding time window (e.g., last 1000 timesteps)
- target_firing_rate is the desired average activity level

Homeostatic Goal: This mechanism maintains stable activity levels across the network. Overactive blocks become harder to excite, preventing runaway excitation. Underactive blocks become easier to excite, preventing silent neurons.

Sensitization and Habituation:

- **Sensitization:** Novel or important stimuli can temporarily lower thresholds, making blocks more responsive. This can be triggered by specific patterns or external signals.
- **Habituation:** Repeated identical stimuli gradually raise thresholds for the responding blocks, reducing response over time.

Threshold Bounds: Thresholds are bounded to maintain functionality: minimum_threshold ≤ threshold ≤ maximum_threshold

Typical bounds might be 30 to 80 on the activation axis, ensuring blocks can fire but require meaningful input.

Dye Interaction: Threshold adaptation can be modulated by dye presence. In dye-rich regions during learning, threshold adaptation may be accelerated or modified to favor successful activity patterns.

6.3 Lag Position Adaptation (Future Work)

Lag position adaptation based on temporal credit assignment is noted as a future extension. For the current addition task implementation, lag positions remain fixed at their initialized values based on shelf structure.

The principle for future implementation: blocks that fire at optimal times relative to task success would adjust their lag positions to maintain that timing, while blocks firing at suboptimal times would shift to better temporal positions. Specific mechanisms for determining optimal timing in the addition task require further development.

6.4 Training Protocol and Memory Formation

6.4.1 Learning Cycle

The system learns through a cyclical process combining computation, evaluation, dye signaling, and consolidation:

Phase 1: Computation

1. Input pattern presented (sensory blocks positioned)
2. System evolves according to dynamics (blocks move, fire, signals propagate)
3. Output emerges (motor block positions)
4. Duration: T timesteps (sufficient for signals to propagate through network)

Phase 2: Evaluation 5. Output is evaluated against desired/correct output 6. Success or failure determined 7. Success metric: similarity between actual and desired output

Phase 3: Dye Release 8. If success: Red dye injected globally or near active regions 9. If failure: No dye injected (or blue dye if using error signaling) 10. Dye release is instantaneous

Phase 4: Consolidation 11. Same or similar input pattern presented again 12. System operates with dye-enhanced wires 13. Hebbian learning active during enhanced state 14. Repeated presentations while dye persists 15. Permanent structural changes accumulate

Phase 5: Decay and Iteration 16. Dye gradually decays 17. System returns to baseline 18. Permanent changes remain 19. Next training pattern presented 20. Cycle repeats

6.4.2 Dye Release Strategies

Spatial Targeting:

Global Release: Dye injected uniformly across entire system. Simpler implementation. All active pathways enhanced equally.

Activity-Proportional Release: Dye injected at locations of recently active blocks. Blocks that fired within temporal eligibility window receive dye at their spatial positions. Implements implicit credit assignment based on participation.

Temporal Specification:

Dye released immediately after successful output detection. Marks pathways that were recently active. Eligibility trace: activity in last N timesteps is eligible for enhancement (typically N = 50-200 timesteps).

6.4.3 Memory Encoding

Memory as Parameter Configuration: A learned pattern is encoded in the specific configuration of wire strengths, thresholds, and lag positions that developed through repeated dye-gated learning.

Distributed Representation: Single memories involve many wires distributed across the network. Single wires participate in multiple memories. Memories are overlapping patterns in parameter space.

Consolidation Process:

1. First exposure: Random or weak response, dye marks success
2. Early trials: Temporary enhancement, variable performance
3. Middle trials: Repeated Hebbian strengthening, improving consistency
4. Late trials: Strong permanent connections, reliable performance
5. After training: Dye absent, structural changes sufficient for recall

Pattern Completion: Partial input activates subset of learned pattern. Strengthened connections drive system toward complete pattern. Output emerges from attractor dynamics in configuration space shaped by learned wire strengths.

6.4.4 Success and Failure Handling

On Success:

- Red dye released
- Active pathways temporarily enhanced
- Hebbian learning amplified
- Successful pattern becomes more strongly encoded

On Failure (Option A: Passive):

- No dye released
- No enhancement
- Hebbian learning at baseline rate (minimal)
- Failed pattern does not consolidate
- Gradually weakens through decay

On Failure (Option B: Active Unlearning):

- Blue dye released
- Active inhibitory pathways enhanced
- OR: Anti-Hebbian rule active (weaken co-active connections)
- Failed pattern actively suppressed
- Faster unlearning

Recommended: Start with Option A, add Option B if needed for faster training.

6.4.5 Training Convergence

Criterion: Training continues until performance reaches threshold (e.g., 95% correct) on training set or validation set.

Overtraining Prevention: Monitor performance on held-out validation data. Stop training if validation performance plateaus or decreases while training performance improves (overfitting).

Plasticity Scheduling: Learning rates and dye enhancement factors can be reduced over training to stabilize learned patterns and prevent catastrophic forgetting.

---

7. Input/Output Interface

7.1 Input Mechanism

The system receives input through sensory blocks that are directly influenced by external signals, typically encoded as pixel arrays.

7.1.1 Screen-to-Block Mapping

Pixel Array: Input is provided as a 2D array of pixels, where each pixel has a brightness or activation value. For binary input, pixels are either ON (1) or OFF (0). For grayscale input, pixels have values in range [0, 1] or [0, 255].

One-to-One Mapping: Each input pixel corresponds to one sensory block. For an input screen of dimensions W × H pixels, the system has W × H sensory blocks designated as input blocks.

Sensory Block Properties: Input blocks are typically red (excitatory) blocks positioned at the left boundary of the activation axis and distributed across various lag positions. Each sensory block has a threshold and can fire like any other block, but receives direct external forcing.

7.1.2 Encoding Scheme

Position-Based Encoding: The brightness of an input pixel determines how far the corresponding sensory block is pushed along the activation axis.

For pixel value p (normalized to [0, 1]): sensory_block_position = origin + (p × input_scaling_factor)

Where input_scaling_factor determines the range of positions input can produce. Typical value: 50-100, meaning full brightness pushes block well past typical thresholds.

Binary Encoding:

- Pixel OFF (0): sensory block remains at origin (position = 0)
- Pixel ON (1): sensory block pushed to position = input_scaling_factor

Grayscale Encoding:

- Dark pixels (low values): small rightward displacement
- Bright pixels (high values): large rightward displacement
- Continuous range allows analog input encoding

7.1.3 Force Application

Direct Positioning: In the simplest implementation, input directly sets sensory block positions at the start of each computation:

At t = 0: For each sensory block i with corresponding pixel value p_i: x_i = origin + (p_i × input_scaling_factor) v_i = 0

Force-Based Input (Alternative): Input can be implemented as external forces applied to sensory blocks:

F_input_i = k_input × (target_position - current_position)

This creates smoother dynamics where input gradually pushes blocks toward target positions rather than instantaneously setting them.

Sustained vs. Transient Input:

- **Sustained:** Input force/position maintained throughout computation
- **Transient:** Input applied briefly, then removed, allowing system to evolve freely

Choice depends on task requirements.

7.1.4 Spatial Arrangement

Sensory blocks can be arranged to preserve input topology:

Grid Layout: For 2D image input, sensory blocks positioned with lag values corresponding to pixel row/column, creating 2D spatial structure in lag-activation space.

Sequential Layout: Blocks arranged in sequence along lag axis, creating temporal processing structure.

Random Layout: Lag positions assigned randomly, requiring system to learn relevant spatial relationships.

The arrangement affects how temporal dynamics interact with spatial input structure.

7.2 Output Mechanism

The system produces output through motor blocks whose positions are read and decoded to produce output values.

7.2.1 Block-to-Screen Mapping

Motor Blocks: A subset of blocks designated as output/motor blocks. These blocks are typically positioned at the right side of the network (receiving signals from hidden layers) and distributed across lag positions.

One-to-One Mapping: Each output pixel corresponds to one motor block. For an output screen of dimensions W_out × H_out, there are W_out × H_out motor blocks.

Output Block Properties: Motor blocks can be any color but are often red (excitatory). They participate in network dynamics like any block but their final positions are read as output.

7.2.2 Decoding Scheme

Position-Based Decoding: The position of a motor block on the activation axis determines the corresponding output pixel value.

For motor block at position x_motor: pixel_value = (x_motor - origin) / output_scaling_factor

Clamped to valid range [0, 1] or [0, 255].

Binary Decoding (Threshold-Based):

- If x_motor > output_threshold: pixel = ON (1)
- If x_motor ≤ output_threshold: pixel = OFF (0)

Output threshold typically set near block threshold values (e.g., 50).

Grayscale Decoding:

- Large positive position → bright pixel
- Small positive position → dim pixel
- Near-origin position → dark/off pixel
- Negative position → clamped to zero (dark)

7.2.3 Readout Timing

Fixed Duration: System evolves for fixed duration T_compute timesteps. Output read at t = T_compute.

Convergence-Based: System evolves until reaching stable state (positions change less than threshold for N consecutive timesteps). Output read upon convergence.

Continuous Readout: Output read at every timestep, providing temporal trajectory of output values. Final output is value at designated readout time.

Multiple Readouts: For temporal sequence tasks, output can be read at multiple time points, producing sequence of outputs from single input.

7.2.4 Spatial Arrangement

Motor blocks arranged to produce meaningful output structure:

Grid Layout: Blocks positioned to create 2D output image structure.

Sequential Layout: Blocks arranged temporally for sequence output or classification (e.g., 10 motor blocks for 10 digit classes, highest activation determines class).

Hierarchical: Groups of motor blocks for structured output (e.g., object class + bounding box coordinates).

7.3 Input-Output Examples

7.3.1 Binary Classification

Task: Determine if input pattern is "X" or "O"

Input:

- 9 pixels (3×3 grid)
- Each pixel ON or OFF
- 9 sensory blocks encode pattern

Output:

- 2 motor blocks (one for "X", one for "O")
- Block with higher position determines classification
- OR: 1 motor block, position > threshold = "X", position ≤ threshold = "O"

7.3.2 Digit Recognition

Task: Recognize handwritten digits 0-9

Input:

- 784 pixels (28×28 grayscale image)
- 784 sensory blocks encode image
- Pixel brightness → block position

Output:

- 10 motor blocks (one per digit class)
- Motor block with highest position indicates predicted digit
- Read output after T_compute timesteps
- Argmax(motor_block_positions) = predicted_digit

7.3.3 Simple Arithmetic

Task: Add two small numbers

Input:

- 10 pixels: first 5 encode first number (binary), second 5 encode second number
- 10 sensory blocks
- Number N represented by N pixels ON

Output:

- 15 pixels (maximum sum is 10, but allow range)
- 15 motor blocks
- Number of motor blocks above threshold = sum
- Count(x_motor > threshold) = output_sum

7.3.4 Temporal Sequence

Task: Predict next element in sequence

Input:

- Sequence of patterns presented over time
- Each pattern: sensory blocks positioned, system evolves briefly
- Next pattern presented, system continues evolving
- Temporal structure encoded through lag positions

Output:

- Motor blocks encode predicted next pattern
- Readout after sequence presentation complete
- Compare prediction to actual next element

7.4 Preprocessing and Postprocessing

Input Preprocessing:

- Normalization: Scale pixel values to standard range
- Noise injection: Add variability for robustness training
- Augmentation: Shifted, rotated, or distorted versions of patterns

Output Postprocessing:

- Thresholding: Convert continuous output to discrete categories
- Smoothing: Average over multiple readouts to reduce noise
- Winner-take-all: Select single highest motor block for classification

7.5 Initialization

Before Computation:

1. Reset all blocks to resting state (position ≈ origin, velocity = 0)
2. Clear refractory timers
3. Apply input (position sensory blocks according to input pattern)
4. Begin dynamics evolution

Between Trials: Option A (Full Reset): Return entire system to resting state Option B (Partial Reset): Reset only sensory and motor blocks, allow hidden blocks to settle naturally Option C (No Reset): Continuous operation, new input applied to current state

Choice depends on whether system should maintain context across inputs (recurrent processing) or treat each input independently (feedforward processing).

---

8. Example Implementation: Single-Digit Addition Network

This section specifies a complete network architecture for a concrete computational task: adding two single-digit numbers (0-9) to produce their sum (0-18).

8.1 Task Specification

Input: Two integers in range [0, 9]

Output: One integer representing their sum, in range [0, 18]

Examples:

- Input: (3, 5) → Output: 8
- Input: (9, 9) → Output: 18
- Input: (0, 7) → Output: 7

8.2 Network Architecture

Total Blocks: 79

Shelf Structure: Four shelves arranged along the lag axis with explicit separation.

Shelf 1 (Input Layer):

- Position: lag 45-55 (center at 50)
- Number of blocks: 20
    - Blocks 1-10: encode first digit
    - Blocks 11-20: encode second digit
- Lag distribution: blocks evenly spaced across range (45, 45.5, 46, 46.5, ..., 54.5, 55)
- Color: all red (excitatory)
- Thresholds: random in range [40, 50]

Shelf 2 (Hidden Layer 1):

- Position: lag 95-105 (center at 100)
- Number of blocks: 20 (blocks 21-40)
- Lag distribution: blocks evenly spaced across range
- Color distribution:
    - 14 red blocks (70%)
    - 5 blue blocks (25%)
    - 1 yellow block (5%)
    - Colors randomly assigned to block positions
- Thresholds: random in range [45, 60]

Shelf 3 (Hidden Layer 2):

- Position: lag 145-155 (center at 150)
- Number of blocks: 20 (blocks 41-60)
- Lag distribution: blocks evenly spaced across range
- Color distribution:
    - 14 red blocks (70%)
    - 5 blue blocks (25%)
    - 1 yellow block (5%)
    - Colors randomly assigned to block positions
- Thresholds: random in range [45, 60]

Shelf 4 (Output Layer):

- Position: lag 195-205 (center at 200)
- Number of blocks: 19 (blocks 61-79)
    - Block 61 represents sum = 0
    - Block 62 represents sum = 1
    - ...
    - Block 79 represents sum = 18
- Lag distribution: blocks evenly spaced across range
- Color: all red (excitatory)

- Thresholds: random in range [50, 60]

8.3 Connectivity

Connection Algorithm:

For each block i:

1. **Local Connections (nearest neighbors):**
    - Identify K=20 blocks with smallest lag distance to block i
    - Create wire from block i to each of these neighbors
    - Wire color matches source block (block i) color
    - Wire strength: random value in range [0.8, 2.5]
2. **Long-Range Connections:**
    - Select L=5 random blocks where |lag_target - lag_i| > 50
    - Create wire from block i to each of these distant blocks
    - Wire color matches source block color
    - Wire strength: random value in range [0.8, 2.5]
3. **Total connections per block:** Approximately 25 (20 local + 5 long-range)
4. **Total connections in network:** Approximately 1,975

Output Layer Lateral Inhibition:

Additionally, for output blocks (blocks 61-79):

Each output block i creates inhibitory connections to nearby output blocks:
```
  for each output block j where |i - j| <= 2:
    create blue wire from block i to block j
    wire strength = 1.8 (inhibitory)
```

This creates competitive dynamics at the output layer. When an output block fires, it inhibits its neighbors, implementing winner-take-all behavior that sharpens the output distribution. This ensures that one output block dominates rather than multiple nearby blocks firing simultaneously.

Expected Connection Pattern:

Due to the increased connection density and shelf structure:

- Most connections (~70-75%) are within the same shelf, creating dense local processing

- Some connections (~20-25%) reach to adjacent shelves, enabling feed-forward and feedback pathways
- Few connections (~5-10%) are long-range across multiple shelves, providing global integration

Information Mixing:

With 20 local connections per input block and 20 blocks in the first hidden shelf:

- Each input block connects to 20 of 20 possible hidden blocks in Shelf 2
- High probability that any hidden block receives from both input groups
- Approximately 90-95% of Shelf 2 hidden blocks will receive connections from both first-digit and second-digit input blocks
- This ensures information from both inputs can be combined for computing sums

The small-world network topology with increased connection density provides both local processing efficiency and global integration capability necessary for the addition task.

8.4 Input Encoding

One-Hot Encoding per Digit:

To represent a digit d (where $0 \leq d \leq 9$):

- Push block (d+1) in the corresponding group to position = 100
- All other blocks in that group remain at position = 0

Example - Input (7, 5):

- First digit = 7:
    - Block 8 (index 7+1) pushed to position 100
    - Blocks 1-7, 9-10 remain at position 0
- Second digit = 5:
    - Block 16 (index 5+1 in second group) pushed to position 100
    - Blocks 11-15, 17-20 remain at position 0

Special Case - Zero:

- Zero is represented by block 1 (first digit) or block 11 (second digit)
- Ensures all digits including zero have dedicated representation

8.5 Output Decoding

Winner-Take-All:

After computation completes (system runs for T=500 timesteps):

1. Read positions of all output blocks (blocks 61-79)
2. Find block with maximum position: winner = argmax(x_61, x_62, ..., x_79)
3. Map winner to sum value:
    - If winner = block 61 → output = 0
    - If winner = block 62 → output = 1
    - ...
    - If winner = block 79 → output = 18

Alternative - Threshold-Based:

Count number of output blocks with position > threshold (e.g., 60):

- Output = count(x_i > 60 for i in [61, 79])

This allows for more distributed output representation but is less precise.

Winner-take-all is recommended for clarity.

8.6 Training Protocol

Dataset:

- All 100 possible input pairs: (0,0), (0,1), ..., (9,9)
- Each pair labeled with correct sum

Training Loop:

```
For epoch in 1 to N_epochs:
   For each (digit1, digit2) in shuffled dataset:

      # 1. Reset system
      Set all blocks to position=0, velocity=0, refractory_timer=0
      Clear signal queue

      # 2. Apply input
      Encode (digit1, digit2) using one-hot scheme

      # 3. Run dynamics
      Evolve system for T=500 timesteps

      # 4. Read output
      winner = argmax(output_block_positions)
      predicted_sum = winner - 60  # Convert block index to sum value
      correct_sum = digit1 + digit2

      # 5. Evaluate
```

```
if predicted_sum == correct_sum:
    success = True
else:
    success = False

# 6. Release dye
if success:
    Inject red dye globally or near active blocks
    Concentration = 0.8

# 7. Consolidation (run again with dye present)
Reset blocks
Encode same (digit1, digit2) input
Evolve for T=500 timesteps
Hebbian learning active (amplified by dye)

# 8. Dye decay
Dye decays over next few trials (τ = 200 timesteps)
```

Success Criterion:

- Training complete when accuracy > 95% on all 100 examples
- Typically requires 1000-5000 training epochs (100,000-500,000 individual trials)

Parameter Updates During Training:

- Wire strengths: continuous Hebbian updates (amplified by dye)
- Thresholds: slow homeostatic adjustment (every 1000 timesteps)
- Lag positions: remain fixed (future work)

8.7 Expected Behavior

Initial (Untrained) State:

- Random wire strengths → chaotic activity
- Unpredictable output
- Accuracy ≈ 5% (random guessing among 19 outputs)

Early Training (100-500 epochs):

- Some patterns begin to emerge
- Output less random but still mostly incorrect
- Accuracy ≈ 20-40%

Mid Training (500-2000 epochs):

- Clear pathways developing
- Certain input combinations consistently produce correct output
- Others still incorrect
- Accuracy ≈ 60-80%

Late Training (2000+ epochs):

- Robust pathways established
- Nearly all combinations correct
- Accuracy > 95%

Post-Training:

- Network reliably computes addition
- Can generalize to noisy inputs (partial patterns still activate correct output)
- Memory encoded in pattern of wire strengths

8.8 Analysis and Visualization

After training, the network can be analyzed:

Wire Strength Patterns:

- Identify which connections strengthened most
- Visualize information flow from input → output
- Detect computational motifs (recurring connection patterns)

Block Activity Patterns:

- For each input pair, record which blocks fire
- Identify blocks specialized for specific digits or operations
- Detect population coding (groups of blocks representing values)

Temporal Dynamics:

- Track how activity propagates through shelves over time
- Measure latency from input to output
- Identify critical blocks that are essential for computation

8.9 Generalization Testing

To evaluate whether the network learns a generalizable computational rule versus memorizing specific input-output mappings, a held-out set experiment can be performed.

Experimental Design:

From the 100 possible input combinations, withhold 2 combinations from training:

- Held-out combination 1: (9, 7) → correct sum: 16
- Held-out combination 2: (2, 4) → correct sum: 6

Train the network on the remaining 98 combinations until achieving >95% accuracy on the training set.

Testing Procedure:

After training completes:

1. Reset system to resting state
2. Apply held-out input (9, 7)
3. Evolve system for 500 timesteps
4. Read output via winner-take-all
5. Compare predicted sum to correct sum (16)
6. Repeat for held-out input (2, 4)

Possible Outcomes:

Outcome A - Successful Generalization: Both held-out combinations produce correct outputs. This indicates the network has learned the computational structure of addition rather than memorizing individual examples. The wire strength patterns encode a generalizable operation that can be applied to novel inputs.

Outcome B - Partial Generalization: Outputs are close but not exact. For example, (9,7) produces 15 or 17 instead of 16. This suggests the network has learned approximate relationships and interpolates between trained neighbors but lacks precision for untrained regions.

Outcome C - Failed Generalization: Outputs are incorrect and not close to correct values. This indicates the network memorized specific input-output pairs without extracting the underlying computational rule. Each trained combination uses dedicated pathways that do not compose to handle novel inputs.

Analysis Methods:

Activity Pattern Comparison: Record which blocks fire for the held-out combination (9,7) and compare to activity patterns for trained neighbors:

- (9,6) → sum 15
- (8,7) → sum 15
- (9,8) → sum 17

If activity patterns for (9,7) are intermediate between these neighbors, this suggests smooth interpolation and compositional representation.

Error Distance: Measure how far the incorrect prediction is from the correct answer. Errors of ±1 indicate near-miss generalization. Large errors indicate complete failure.

Retraining Speed: After testing, train the network on the held-out combinations. If it learns them quickly (10-50 epochs), the network was "almost there" and needed minimal adjustment. If it requires extensive training (500+ epochs), the network had no relevant structure for these combinations.

Factors Affecting Generalization:

Several architectural and training factors influence generalization capability:

Network Capacity: Larger networks (more blocks, more connections) can memorize more easily without needing to compress information into generalizable patterns. Smaller networks are forced to find compact representations and may generalize better.

Dye Diffusion: Wider dye spread during training affects larger regions, creating broader learning updates that may lead to more generalizable representations. Narrow dye concentration creates specific, localized learning.

Training Duration: Excessive training can lead to overfitting where the network increasingly specializes to the training set. Stopping training shortly after achieving accuracy threshold may preserve generalization.

Connection Topology: The small-world structure with local clustering and long-range connections should support generalization by providing multiple pathways and redundant representations.

Variations:

Different held-out sets can test various aspects of generalization:

Edge Cases: Hold out (9,9) and (0,0) - extreme values that test extrapolation beyond typical training examples.

Central Cases: Hold out (5,5) and (4,4) - well-surrounded by training examples, testing interpolation in dense regions.

Systematic Patterns: Hold out all combinations where sum=10, testing whether the network learns sum values as meaningful categories.

This generalization experiment provides insight into whether the Neural-Klotski system develops compositional computational representations or operates as a sophisticated pattern matcher.

---

9. Complete Mathematical Formulation

This section provides the complete mathematical specification of the system dynamics, sufficient for precise implementation.

9.1 State Variables

The complete system state at time t is defined by:

For each block i ∈ {1, 2, ..., N}:

- x_i(t): position on activation axis (scalar, continuous)
- v_i(t): velocity on activation axis (scalar, continuous)
- λ_i: position on lag axis (scalar, fixed during operation, adjustable during learning)
- τ_i: firing threshold (scalar, adjustable during learning)
- c_i: color identity (red, blue, or yellow - fixed)
- r_i(t): refractory timer (non-negative scalar, gates firing ability)

For each wire j connecting source block s_j to target block t_j:

- w_j: strength parameter (scalar, adjustable during learning)
- col_j: color (inherited from source block c_{s_j})
- d_j: accumulated damage (scalar, non-negative)

For dye field:

- C_red(x, y, t): red dye concentration at spatial position (x, y) and time t
- C_blue(x, y, t): blue dye concentration at spatial position (x, y) and time t
- C_yellow(x, y, t): yellow dye concentration at spatial position (x, y) and time t

For signal queue:

- signal_queue: list of pending signals with {wire_index, target_block, arrival_time, strength, color}

System state vector: S(t) = {x_i(t), v_i(t), r_i(t) for all i; d_j for all j; C_red, C_blue, C_yellow; signal_queue}

Plus fixed parameters: {λ_i, τ_i, c_i for all i; w_j, col_j for all j}

9.2 Dynamics Equations

9.2.1 Block Position and Velocity

For all blocks, regardless of refractory state, position and velocity evolve according to:

Position update:

$dx_i/dt = v_i$

Velocity update:

$dv_i/dt = (1/m) \times [F_{spring\_i} + F_{wire\_i} + F_{input\_i}] - \gamma \times v_i$

Where:

- m: effective mass (typically set to 1)
- γ: damping coefficient (typically 0.1-0.2, creates overdamped motion)
- F_{spring_i}: restoring force toward origin
- F_{wire_i}: sum of forces from incoming wire signals
- F_{input_i}: external input force (for sensory blocks only)

Spring force:

$F_{spring\_i} = -k \times x_i$

Where k is spring constant (typically 0.1-0.2)

Wire forces:

$F_{wire\_i} = \Sigma_{j:\, t_j = i} F_{wire\_j}$

Sum over all wires j targeting block i.

For each wire j delivering signal at current timestep:

$w_{eff\_j} = w_j \times (1 + \alpha \times C_{color\_j}(\text{position of wire}))$

If col_j = red:
  $F_{wire\_j} = +w_{eff\_j}$

If col_j = blue:
  $F_{wire\_j} = -w_{eff\_j}$

If col_j = yellow:
  $F_{wire\_j} = w_{eff\_j} \times (x_{s\_j} - x_i)$ [always active]

Where:

- w_eff_j: effective strength including dye enhancement
- α: enhancement factor (typically 1.0-3.0)
- C_color_j: dye concentration of matching color at wire location
- For yellow wires, force is proportional to position difference (always active, no delay)

Signal Queue Management:

```
signal_queue = []

When block s_j fires at time t:
    for each outgoing wire j:
        lag_distance = |λ_{t_j} - λ_{s_j}|
        delay = lag_distance / v_signal

        signal_queue.append({
            wire_index: j,
            target_block: t_j,
            arrival_time: t + delay,
            strength: w_j,
            color: col_j
        })

Each timestep:
    current_signals = [sig for sig in signal_queue if sig.arrival_time == current_time]

    for signal in current_signals:
        apply_force_from_signal(signal)

    signal_queue = [sig for sig in signal_queue if sig.arrival_time > current_time]
```

Where v_signal is the signal propagation speed (typically 50-200 lag units per timestep).

Discrete time integration (for simulation):

$$x_i(t + \Delta t) = x_i(t) + v_i(t) \times \Delta t$$

$$v_i(t + \Delta t) = v_i(t) \times (1 - \gamma \times \Delta t) + (F_{total\_i} / m) \times \Delta t$$

Where Δt is the timestep size (typically 0.1-1.0).

9.2.2 Threshold Crossing and Firing

At each timestep, check for threshold crossing **from below**:

```
If x_i(t-Δt) < τ_i AND x_i(t) ≥ τ_i AND r_i(t) = 0:
   # Block fires (must cross threshold from below)
   # Impart refractory kick (negative velocity)
   v_i ← v_i - v_kick

   # Start refractory timer
   r_i ← T_refrac

   # Generate signals on all outgoing wires
   for each outgoing wire j from block i:
      add signal to signal_queue with appropriate delay

   # Mark that this block fired (for learning)
   fired_this_step[i] = True
Else:
   fired_this_step[i] = False
```

Critical requirement: The block must cross the threshold **from below** (position was below threshold in previous timestep and is at or above threshold in current timestep). A block that is already above threshold, or that approaches threshold from above, does not fire.

This requirement:

- Prevents continuous firing while position remains above threshold
- Ensures firing is a discrete event triggered by upward crossing
- Matches biological neurons where sodium channel inactivation prevents repeated firing without returning to resting potential
- Requires block to drop below threshold (during refractory return) before it can fire again

Where:

- v_kick: refractory kick magnitude (typically 15-25, imparts leftward velocity)
- T_refrac: refractory period duration (typically 20-50 timesteps)
- fired_this_step: computed flag array, not a state variable

Refractory behavior: After receiving the velocity kick, the block naturally moves leftward due to the imparted negative velocity combined with spring and damping forces. The block typically reaches a minimum position (around -20 to -40) before reversing direction and returning toward origin. During this entire period, incoming wire forces continue to affect the block's motion. Strong excitatory input can push the block back rightward even during refractory, potentially positioning it near threshold by the time the refractory timer expires. However, the block must cross the threshold from below to fire again, ensuring proper firing dynamics.

9.2.3 Refractory Timer

For all blocks:

If $r_i(t) > 0$:
$r_i(t + \Delta t) = r_i(t) - \Delta t$

Threshold crossing detection is disabled
(even if x_i crosses τ_i, block does not fire)

Position and velocity still evolve normally according to dynamics

The refractory timer gates firing ability but does not directly control position. When $r_i > 0$, the block cannot fire regardless of its position relative to threshold.

9.3 Dye Dynamics

9.3.1 Diffusion Equation

For each color $c \in \{red, blue, yellow\}$:

$\partial C_c/\partial t = D \times \nabla^2 C_c - C_c/\tau_{decay}$

Where:

- D: diffusion coefficient (typically 0.5-2.0)
- τ_{decay}: decay time constant (typically 100-1000 timesteps)
- $\nabla^2 C_c$: spatial Laplacian (second derivative in space)

Discrete approximation (2D grid):

$$C_c(x, y, t + \Delta t) = C_c(x, y, t) + D \times \Delta t \times [C_c(x+\Delta x,y,t) + C_c(x-\Delta x,y,t) + C_c(x,y+\Delta y,t) + C_c(x,y-\Delta y,t) - 4\times C_c(x,y,t)] / (\Delta x^2) - (\Delta t / \tau_{decay}) \times C_c(x, y, t)$$

9.3.2 Dye Injection

Dye injection occurs after trial evaluation based on success or failure.

Spatial targeting: Dye is injected in the spatial region near output/motor blocks. The specific injection locations correspond to the positions of blocks that contributed to the output.

Temporal eligibility: Only wires whose source blocks fired within a recent temporal window (last T_window timesteps, typically 50-200) are eligible for enhancement.

Injection procedure:

```
If trial successful:
    # Identify recently active wires
    eligible_wires = [wire j where source block s_j fired within last T_window timesteps]

    # Inject dye at spatial locations corresponding to these wires
    for wire j in eligible_wires:
        location = spatial_position_of_wire_j
        color = col_j

        C_color(location, t_inject) ← C_color(location, t_inject) + C_inject

If trial failed:
    # Option: No dye injection (passive - failed patterns not reinforced)
    # Alternative: Inject blue dye to actively suppress failed patterns (future work)
```

Where:

- C_inject: injected concentration amount (typically 0.5-1.0)
- spatial_position_of_wire_j: the location in 2D space where wire j exists, typically near its source or target block on the lag-activation plane

Dye spread and effect: After injection, dye diffuses spatially according to the diffusion equation. Wires in dye-rich regions experience enhanced effective strength during subsequent trials. This temporary enhancement, combined with Hebbian learning, leads to permanent strengthening of successful pathways.

9.4 Plasticity Rules

9.4.1 Wire Strength - Hebbian Learning

During operation, when both source and target blocks show correlated activity:

```
If source block s_j fired AND target block t_j fired within temporal window T_window:

    dye_factor_j = 1 + β × C_{col_j}(location of wire j, current_time)

    Δw_j = η_w × dye_factor_j × activity_correlation
```

Where:
  activity_correlation = 1 (both blocks active)

Update:

  w_j ← w_j + Δw_j

Parameters:

- η_w: learning rate for wire strength (typically 0.001-0.01)
- β: dye amplification factor (typically 2.0-5.0)
- T_window: temporal window for correlation (typically 10-50 timesteps)

Spike-Timing Dependent Plasticity (STDP):

If source fires at t_s and target fires at t_t:
  Δt = t_t - t_s

  If 0 < Δt < T_window (source before target):
    Δw_j = +η_w × dye_factor_j × exp(-Δt / τ_STDP)

  If -T_window < Δt < 0 (target before source):
    Δw_j = -η_w × dye_factor_j × exp(Δt / τ_STDP)

Where τ_STDP is the STDP time constant (typically 10-20 timesteps).

Key mechanism: Hebbian learning occurs continuously, but is dramatically amplified in regions where dye is present. Successful patterns marked by dye undergo strong permanent strengthening through repeated trials while dye concentration remains elevated.

9.4.2 Wire Strength - Fatigue and Recovery

Damage accumulation (each time wire transmits signal):

Δd_j = κ_damage × |w_j|

Recovery (continuous):

dd_j/dt = -κ_repair × d_j

Effective strength:

w_eff_j = w_j × (1 - d_j / d_max)

Parameters:

- κ_damage: damage accumulation rate (typically 0.01-0.1)
- κ_repair: repair rate (typically 0.001-0.01)
- d_max: maximum damage (typically 1.0-5.0)

9.4.3 Threshold Adaptation

Measured over sliding window of T_measure timesteps:

```
firing_rate_i = (number of times block i fired in last T_measure timesteps) / T_measure

If firing_rate_i > firing_rate_target:
   Δτ_i = +η_τ × (firing_rate_i - firing_rate_target)

If firing_rate_i < firing_rate_target:
   Δτ_i = -η_τ × (firing_rate_target - firing_rate_i)
```

Parameters:

- η_τ: threshold adaptation learning rate (typically 0.001-0.01)
- firing_rate_target: target firing rate (typically 0.05-0.2, i.e., 5-20% of timesteps)
- T_measure: measurement window (typically 1000-10000 timesteps)

Bounds:

```
τ_min ≤ τ_i ≤ τ_max
```

Typically: τ_min = 30, τ_max = 80

9.4.4 Lag Position Adaptation (Future Work)

Lag position adaptation based on temporal credit assignment is noted as a future extension. For the current addition task implementation, lag positions remain fixed at their initialized values based on shelf structure.

The principle for future implementation: blocks that fire at optimal times relative to task success would adjust their lag positions to maintain that timing, while blocks firing at suboptimal times would shift to better temporal positions. Specific mechanisms for determining optimal timing in the addition task require further development.

9.5 Input and Output

9.5.1 Input Application

For sensory block i with corresponding input value $p_i \in [0, 1]$:

Direct positioning:

$$x_i(t=0) = p_i \times scale\_input$$

$$v_i(t=0) = 0$$

Where scale_input is typically 80-120.

Force-based (alternative):

$$F\_input_i = k\_input \times (p_i \times scale\_input - x_i)$$

Applied continuously during input presentation period.

9.5.2 Output Readout

At readout time T_compute:

Winner-take-all:

$$output = \text{argmax}_{i \in output\_blocks} x_i(T\_compute)$$

Threshold-based:

$$output = \Sigma_{i \in output\_blocks} [x_i(T\_compute) > \theta\_output]$$

Where [condition] = 1 if true, 0 if false, and θ_output is output threshold (typically 50-60).

9.6 Complete Update Algorithm

Single timestep update ($t \rightarrow t + \Delta t$):

1. For each block i:
   a. Calculate F_spring_i, F_wire_i, F_input_i
   b. Update velocity: $v_i \leftarrow v_i \times (1 - \gamma\Delta t) + (F\_total/m) \times \Delta t$
   c. Update position: $x_i \leftarrow x_i + v_i \times \Delta t$

2. For each block i where $r_i = 0$:
   a. Check threshold crossing: if $x_i(t) < \tau_i$ and $x_i(t+\Delta t) \geq \tau_i$
   b. If crossed: impart velocity kick $v_i \leftarrow v_i - v\_kick$, set $r_i = T\_refrac$

3. For each block i where $r_i > 0$:
   a. Decrement refractory timer: $r_i \leftarrow r_i - \Delta t$

4. For each wire j:
   a. Update damage: $d_j \leftarrow \max(0, d_j - \kappa_{repair} \times \Delta t)$
   b. If source block fired this step: $d_j \leftarrow d_j + \kappa_{damage} \times |w_j|$

5. Update dye field:
   a. Apply diffusion equation for each color
   b. Apply decay: $C_c \leftarrow C_c \times \exp(-\Delta t/\tau_{decay})$

6. Apply plasticity updates (if learning enabled):
   a. Hebbian wire strength updates
   b. Threshold adaptation (periodic, not every timestep)

7. Increment time: $t \leftarrow t + \Delta t$

9.7 Parameter Summary

Typical parameter values for stable operation:

Dynamics:

- Δt: 0.1-1.0 (timestep size)
- m: 1.0 (block mass)
- γ: 0.1-0.2 (damping coefficient)
- k: 0.1-0.2 (spring constant)
- v_{kick}: 15-25 (refractory kick velocity magnitude)
- T_{refrac}: 20-50 (refractory period duration)

Wires:

- w_j initial: 0.8-2.5 (random)
- w_{min}: 0.1, w_{max}: 10.0 (bounds)

Dyes:

- D: 0.5-2.0 (diffusion coefficient)
- τ_{decay}: 100-1000 (decay time constant)
- α: 1.0-3.0 (enhancement factor)
- C_{inject}: 0.5-1.0 (injection amount)
- T_{window}: 50-200 (temporal eligibility window)

Learning:

- η_w: 0.001-0.01 (wire learning rate)
- β: 2.0-5.0 (dye amplification)
- η_τ: 0.001-0.01 (threshold learning rate)

- τ_STDP: 10-20 (STDP time constant)

Thresholds:

- τ_i: 40-60 (typical range)
- τ_min: 30, τ_max: 80 (bounds)

These values provide starting points and should be tuned based on specific task requirements and empirical testing.