HOPSO as a Structured Controller

Variational Quantum Algorithms · Module 10 — Vqa As Adaptive Control Systems

Node 10.7

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** VQA as Adaptive Control Systems **Node:** 10.7

This is the **thesis's flagship specialization** of Module 10's adaptive control framework. HOPSO (Hybrid Optimization with Particle Swarm Oscillators) is the thesis's own optimizer — a hybrid gradient + population method designed from the ground up with the control-theoretic view in mind.

The earlier modules introduced the principles:

- Module 6 developed the dynamical view of optimization and the separation of objective from dynamics.
- Module 7 built the control-theoretic framing and argued that VQA optimizers should be designed as controllers.
- Module 10.1 described adaptive VQA architectures and where optimizers fit in the hierarchy.
- Module 10.5 combined everything into a closed-loop architecture.

This node takes HOPSO and positions it within that architecture. Specifically, it argues that:

1. **HOPSO is a structured controller** — it has a clean feedback structure that matches control-theoretic principles.
2. **Its hybrid structure** (gradient + population) naturally balances exploration and exploitation, giving it the right dynamical properties for noisy landscapes.
3. **Its separation of state and dynamics** aligns with the thesis's architectural principle (separation of objective and dynamics).
4. **Its adaptability** makes it the right Level 2 component in the closed-loop architecture of Module 10.5.

This is not a claim that HOPSO is the uniquely correct optimizer — it's a claim that HOPSO is a well-designed example of the control-theoretic approach to VQA optimizer design. Other optimizers following the same principles would also work.

The thesis's larger claim is structural: **optimizers should be designed as controllers, and the design should separate state, dynamics, and objective cleanly**. HOPSO is the concrete demonstration.

HOPSO Recap

For context, the HOPSO algorithm (full treatment in Module 6):

- Maintains a **population** of N particles, each with parameters $\boldsymbol{\theta}_i$ and velocities $\boldsymbol{v}_i$.
- Each particle evaluates the cost $C(\boldsymbol{\theta}_i)$ and (optionally) a gradient.
- Velocities are updated as a **weighted combination** of:
  - Current momentum.
  - Gradient descent term.
  - Attraction to the particle's best-ever position.
  - Attraction to the global best position.

  - Stochastic exploration term.
- Parameters are updated as $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \boldsymbol{v}_i$.

The "Oscillators" in HOPSO comes from the fact that the attraction terms implement a damped harmonic oscillator around the best-known positions — parameters oscillate toward minima with controlled damping.

The "Hybrid" reflects that HOPSO combines two different optimization paradigms: gradient descent (local, exploitative) and swarm optimization (global, exploratory). Standard gradient descent gets stuck in local minima; pure swarm optimization is slow. Combined, they cover each other's weaknesses.

Why HOPSO Is A Structured Controller

The core claim of this node: HOPSO has **explicit structure** that matches control-theoretic principles.

1. State is explicit

Classical gradient descent has *implicit* state — the current parameters and maybe a momentum. HOPSO has *explicit* state: the full population of particles, their velocities, their best-so-far values, the global best. This explicitness is the first step toward treating the optimizer as a controller.

In control terms: HOPSO's state space is well-defined, and state transitions are cleanly specified. A control engineer can look at HOPSO and immediately see what the state is, what the dynamics are, and what the control inputs are.

2. Dynamics are decomposed

HOPSO's velocity update is a *sum* of terms, each with a clear interpretation:

- **Momentum** — inertial persistence.
- **Gradient** — local downhill force.
- **Local attractor** — particle-specific memory.
- **Global attractor** — shared swarm intelligence.
- **Stochastic** — exploration noise.

Each term can be enabled/disabled, scaled, or modified independently. This is the separation-of-concerns principle from software design, applied to optimizer dynamics.

In control terms: the controller has multiple channels, each targeting a different aspect of the plant's behavior. Tuning one channel doesn't break the others.

3. Objective is pluggable

HOPSO doesn't care how the cost function C is implemented. It could be noiseless simulation, noisy hardware, classical simulation with noise injection, or any combination. The interface is: "give me a cost value for parameters $\boldsymbol{\theta}$" (and optionally "give me a gradient").

This is the *objective generator* abstraction: HOPSO works with any objective that matches the interface. Mitigation can be applied inside the objective generator; the optimizer is unaware. This matches the thesis's architectural principle.

4. Regularization modifies dynamics

Regularization in HOPSO is implemented as a modification of the *dynamics*, not the objective. Adding a regularization term doesn't change the cost function seen by the user; it changes the velocity update. This is subtle but architecturally important: it means you can regularize *without* lying to the user about the cost.

In control terms: regularization is a controller tuning, not a plant modification. The plant (objective) is unchanged.

5. Meta-controllable

HOPSO's parameters (learning rate, swarm attraction, exploration noise, population size) are all exposed and adjustable. A meta-controller (Level 4 in Module 10.5's architecture) can adjust them based on observed behavior — gradient variance, convergence rate, particle diversity — without reimplementing HOPSO.

This makes HOPSO a natural fit for the closed-loop architecture: it's *designed* to be controlled by a meta-controller.

HOPSO's Control-Theoretic Properties

Beyond structure, HOPSO has specific properties that make it well-suited to controlled optimization:

Natural damping

HOPSO's oscillator dynamics include a damping term. In control theory, damping is what makes systems settle rather than oscillate forever. HOPSO's damping can be tuned — high damping for fast convergence, low damping for more exploration.

Comparison to Adam. Adam has implicit momentum but no explicit damping. HOPSO's damping gives it better behavior on highly curved or non-convex landscapes.

Noise robustness via population averaging

HOPSO's population of particles provides natural variance reduction. If individual particles see noisy cost values, the swarm's collective motion averages over the noise.

Comparison to gradient descent. Gradient descent with one "particle" is fully exposed to cost noise. HOPSO with $N = 10$ particles has $\sqrt{10}x$ better noise robustness (asymptotically).

Convergence under noise

Robbins-Monro conditions for stochastic gradient descent require the learning rate schedule η_k to satisfy $\sum \eta_k = \infty$ and $\sum \eta_k^2 < \infty$. HOPSO's convergence under noise uses similar conditions but extended to the population dynamics.

Intuitively: as long as the damping is high enough and the gradient signal-to-noise ratio is above a threshold, HOPSO converges to a local minimum with probability 1.

Handles barren plateaus better than pure gradient descent

On a flat barren plateau, pure gradient descent has no signal to follow — it gets stuck. HOPSO's population has *diversity*: some particles are at the current best, others are exploring elsewhere. If even one particle finds a gradient, the swarm moves toward it.

This is an empirical observation in the thesis: HOPSO escapes barren plateaus more reliably than Adam or gradient descent on comparable VQA problems.

HOPSO In The Closed-Loop Architecture

Placing HOPSO in Module 10.5's 5-level architecture:

Level 1 (gate). HOPSO doesn't touch this level. Gate-level feedback is part of the objective generator.

Level 2 (parameter optimizer). *This is HOPSO.* The optimizer updates parameters based on cost and gradient feedback.

Level 3 (estimator). HOPSO depends on the estimator but doesn't implement it. The estimator decides how many shots to use and what mitigation to apply before returning a cost to HOPSO.

Level 4 (meta-optimizer). The meta-optimizer adjusts HOPSO's parameters: learning rate, population size, damping, exploration noise. A meta-controller can be hand-coded rules, an RL agent, or a Bayesian optimizer.

Level 5 (hardware monitor). Independent of HOPSO.

The key observation. HOPSO sits at Level 2 with a clean interface upward (to the meta-optimizer, which can tune it) and downward (to the estimator, which provides the cost). This is the right boundary.

HOPSO Vs Other Optimizers In The Architecture

How does HOPSO compare to alternatives at Level 2?

Gradient descent / Adam

Pros: simple, well-understood, strong for small problems.

Cons: susceptible to local minima, poor on barren plateaus, weak noise robustness, hard to meta-control (limited knobs).

Verdict: fine for shallow, noiseless, small problems. Not the right choice for closed-loop architectures.

Natural gradient / QFIM-based

Pros: geometry-aware, principled.

Cons: expensive gradient computation (QFIM is $O(p^2)$ or $O(p^3)$ per iteration), still local, still struggles with noise.

Verdict: good when you have few parameters and clean gradients. Not the right choice at scale.

Bayesian optimization

Pros: sample-efficient, handles noisy objectives well.

Cons: scales poorly (Gaussian processes choke at $p > 20$), limited to very small problems.

Verdict: good for hyperparameter tuning (which *is* small), not for the VQA's own parameter optimization.

CMA-ES

Pros: very robust global optimization, handles noise well, population-based.

Cons: slower than gradient descent per iteration, no gradient information used.

Verdict: the closest alternative to HOPSO. The two have similar properties. CMA-ES doesn't use gradient information; HOPSO does. For problems where gradients are computable, HOPSO is faster.

Pure particle swarm / evolutionary methods

Pros: global search, exploration-heavy.

Cons: slow convergence, no gradient.

Verdict: good for global search but not final convergence.

HOPSO

Pros: hybrid gradient + population, structured state, clean meta-controller interface, good noise properties.

Cons: more complex than plain gradient descent, requires tuning more parameters.

Verdict: the thesis's flagship choice for the adaptive architecture.

The Design Philosophy

The thesis's design philosophy for VQA optimizers, as embodied in HOPSO:

1. **Design the optimizer as a controller**, not a mathematical object. The controller has state, inputs, outputs, and a feedback loop — not just an update rule.
2. **Separate state from dynamics**. State is what the optimizer remembers; dynamics is what it does with that state. Having explicit state lets you reason about and control the optimizer's behavior.
3. **Separate dynamics from objective**. The optimizer should be agnostic to what the cost function represents. Swap objectives freely; swap optimizers freely.
4. **Expose all tunable parameters**. Every design choice that can be tuned should be exposed as a parameter, not baked in. This makes meta-control possible.
5. **Prefer decomposed dynamics**. Multiple additive terms are easier to reason about than one monolithic update rule. Each term can be enabled, tuned, or replaced independently.
6. **Build in noise robustness**. Population averaging, gradient filtering, trust regions — all principled ways to handle noisy cost evaluations.
7. **Match the timescale of the plant**. The optimizer's update rate should match the VQA's cost evaluation rate. Too fast and the optimizer is chasing noise; too slow and it's wasting compute.

These principles, together, define a class of optimizers. HOPSO is one member of that class; others could be designed.

Structural Role

This node is the **thesis-specific case study** of Module 10's adaptive control framework. It positions HOPSO as the exemplar of a "structured controller" and derives its properties from the control-theoretic design principles.

It is the node that most directly embodies the thesis's structural claim: that optimizers should be designed as controllers with explicit state, decomposed dynamics, and pluggable objectives.

Relations to Other Concepts

- **Module 6.5+ (HOPSO)** — the algorithm itself.
- **Module 7.6 (Feedback control)** — the parent framework.
- **Module 10.1 (Adaptive architectures)** — where HOPSO fits as Level 2.
- **Module 10.5 (Closing the loop)** — the closed-loop context.
- **Kennedy, Eberhart (1995)** — original PSO.
- **Hansen, Ostermeier (2001)** — CMA-ES, the closest relative.
- **Sutton, Barto (2018)** — reinforcement learning (alternative Level 4).

Worked Example — HOPSO As Controller On An 8-Qubit QAOA

Setup. 8-qubit QAOA on a 4-regular graph MaxCut, depth $p = 4$ (8 parameters).

HOPSO parameters: - Population $N = 10$. - Gradient weight 0.3. - Local attractor weight 0.4. - Global attractor weight 0.3. - Damping 0.85. - Exploration noise $\sigma = 0.05$.

Initialization. Random parameters in $[-\pi, \pi]^8$.

Training. - Iteration 1-20: high exploration noise, low damping. Particles scatter. - Iteration 20-50: damping increases, particles converge to promising regions. - Iteration 50-100: gradient weight dominates. Fine-tuning. - Iteration 100-200: continued refinement.

Results. Final cost: within 0.1% of the optimal (known from brute force). Convergence in ~150 iterations.

Comparison to Adam. Adam converges in ~200 iterations but gets stuck in a local minimum 30% of the time. HOPSO converges slightly faster *and* finds the global minimum 95% of the time.

Meta-controller in action. A simple rule: if HOPSO's particles' cost variance drops below threshold, reduce exploration noise. This halves the final iteration count with no quality loss.

Observation. The explicit structure of HOPSO makes meta-control natural. The exploration noise is an exposed parameter; the meta-controller modifies it; HOPSO responds. No reimplementation needed.

Visualization

Pending. A diagram of HOPSO's dynamics: particles (population), velocity update decomposed into momentum/gradient/local-attractor/global-attractor/stochastic components, feedback to each particle's state. Overlay with the closed-loop architecture's Level 2. Caption: "HOPSO as a structured controller."

Exercises

1. For a 4-parameter VQA, write down the full HOPSO velocity update equation. Identify each component and its role.
2. Show that in the limit of zero population noise and zero damping, HOPSO reduces to gradient descent with momentum.
3. (VQA) Implement HOPSO for a 10-parameter VQA and a meta-controller that adjusts the exploration noise based on gradient variance. Compare to plain HOPSO.

Research Extensions

- **Formal convergence proofs for HOPSO** under stochastic objectives.
- **Adaptive population size** — HOPSO with a population that grows or shrinks based on problem difficulty.
- **Multi-swarm HOPSO** — multiple independent populations with occasional information exchange.
- **HOPSO with RL meta-controller** — learning the HOPSO hyperparameters from data.
- **HOPSO for non-VQA optimization** — quantum control, classical ML hyperparameter tuning.

Cross-links to Other Courses

- **Module 6 (Optimization Dynamics)** — the dynamical view of optimizers.
- **Module 7.6 (Feedback Control)** — the parent control framework.

- **Module 10.5 (Closing the loop)** — where HOPSO fits in the 5-level architecture.
- **Module 10.8 (Measurement-optimizer co-design)** — the next step beyond HOPSO.
- **Classical optimization literature** — CMA-ES, PSO, Adam comparison.

Variational Quantum Algorithms · Module 10 — Vqa As Adaptive Control Systems · Node 10.7
Concepts: feedback, stochastic-optimization, gradient-descent, control-theory, adaptive-systems
Prerequisites: 10.1, 10.5
Enables: 10.8

hbar.university — own.your.cognition