NOVA Autonomous Inspection Robot: An Evidence-Aware Architecture, Prototype Evaluation, and Integration Roadmap

Aaron Singh

***Abstract*—This paper presents NOVA Autonomous Inspection Robot, a project in a twenty-project AI infrastructure portfolio. The current repository is an active prototype with implemented behavior, automated correctness tests, and explicit boundaries around unverified hardware or production claims. We describe the system model, current implementation, goals, and evaluation method, then propose five integrations that advance the project toward reproducible system-level validation. A local audit executed 4 project tests successfully. No accelerator, deployment, or performance conclusion is inferred unless a corresponding committed benchmark or profiler artifact exists.**

***Index Terms*—NOVA, AI infrastructure, reproducibility, prototype validation, systems evaluation, hardware–software co-design**

I. Introduction

NOVA Autonomous Inspection Robot (P15) addresses a bounded problem within the portfolio's track-04-robotics-edge track. Its declared status is *Active Prototype* [1]. The project validates its central mechanism before adding device-specific acceleration, production orchestration, or real-world hardware. This ordering matters because optimization without a trusted reference can make incorrect behavior appear successful.

The project goals are to establish deterministic behavior, encode correctness as tests, create machine-readable evidence, identify limiting resources through measurement, and integrate results with adjacent portfolio systems without losing provenance. The repository contains one paper named exactly after its singular folder: `P15-nova-autonomous-inspection-robot.tex`.

II. Technical Context

Edge and robotic systems operate under resource, latency, privacy, and safety constraints. Development should progress from deterministic simulation to representative sensors, middleware, and hardware-in-the-loop validation. The portfolio benchmark standard requires environment, workload, method, metric, artifact, reproduction, and limitation fields; unknown values remain explicitly unmeasured [3].

III. System Model and Architecture

The prototype is organized around the following domain model:

$$d_1((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2| \qquad (1)$$

The equation is a design and test abstraction rather than a claimed empirical law. It supports invariants and expected-value checks while later implementations replace synthetic inputs with representative workloads or devices.

The software architecture contains an input/configuration layer, a deterministic core, validation and evidence output, and a local visualization. The principal inspected source artifacts are `python/inspection.py`. Unsupported real-world conditions are surfaced as limitations rather than silently simulated.

IV. Detailed Script Operation and Rationale

The inspection script searches a synthetic grid for the brightest defect candidate, handles the no-defect case, and generates a bounded Manhattan path that can stop before contact. It separates perception and path safety logic from cameras, kinematics, collision models, and robot hardware.

The execution path is:

1) Validate and scan the synthetic inspection grid.
2) Select the brightest cell above the defect threshold.
3) Return an explicit no-defect outcome when appropriate.
4) Construct a Manhattan path while enforcing grid bounds.
5) Stop at the target or before contact according to the safety flag.

V. Implemented Prototype

The metadata and source audit found these completed features [2]:

- Synthetic grid defect detection
- Bounded Manhattan path
- No-defect and out-of-bounds tests
- Stop-before-contact flag

`python3 -m unittest discover -s tests -p 'test_*.py'` completed successfully with 4 tests. The inspected test artifacts are `tests/test_inspection.py`. This is evidence of local correctness for encoded cases, not production scale or hardware performance.

VI. Testing Methodology and Observed Results

Testing uses Python's standard `unittest` discovery and exercises the public behavior of the reference implementation. The audit reran the suite from the project folder with `python3 -m unittest discover -s tests -p 'test_*.py'`. All 4 discovered tests passed. The result establishes correctness only for the encoded

TABLE I
IMPLEMENTATION ARTIFACTS AND WHY THEY EXIST

Artifact	Observed responsibility	Engineering rationale
python/inspection.py	find_defect, path, main	Implements the inspectable, unit-tested project core.
scripts/reproduce.sh	Fixed test and demonstration entry point	Gives another developer one command for local reproduction.
PROJECT.yaml and ANALYSIS.md	Status, completed work, planned work, and claim boundaries	Separates declared intent from evidence-backed implementation.
streamlit_app.py	Local evidence and status visualization	Makes outputs inspectable without upgrading simulation into a hardware claim.

local cases; it does not establish accelerator correctness, real-device behavior, production reliability, or benchmark completion.

No numerical performance result is promoted by this test run. Where scripts emit JSON or JSONL, those outputs remain raw or simulation-specific until a reviewed summary includes hardware, software, workload, warm-up, repetition, correctness threshold, Git revision, and limitations.

VII. Claim Boundaries and Risks

The project records these unverified or excluded claims:

- No camera, physical robot, or autonomous hardware inspection

The main risk is confusing synthetic or modeled behavior with deployed-system behavior. Other risks include incomplete workloads, platform-dependent timing, missing failure injection, and interfaces not yet exercised across device boundaries. Performance claims require a reviewed record meeting the portfolio standard.

VIII. Evaluation Plan

Evaluation proceeds through correctness tests, deterministic reproduction with Git and environment metadata, repeated benchmarks reporting latency/throughput/memory/error metrics, and a named profiler capture tied to exact hardware and source revision. Success requires reference equivalence within a documented tolerance, preservation of safety and resource invariants, clear failures, and evidence reproducible from a clean environment.

IX. Goals, Milestones, and Success Criteria

The project goals are staged so that correctness precedes performance and integration. A goal is complete only when its proof artifact is committed or otherwise reviewable; prose or a simulated number alone is insufficient.

X. Future Work and Integrations

The five project-specific next steps are:

1) Image dataset baseline
2) Kinematics simulator
3) Collision and hardware safety
4) Add perception uncertainty, collision avoidance, and supervised recovery.
5) Integrate P14 control, P13 routing, and P12 reliability reporting.

The early items complete declared evidence; the later items connect downstream portfolio consumers. Each integration should add tests and a reviewable artifact such as JSONL evidence, a report, profiler capture, deployment manifest, trace, or labeled data set.

XI. Conclusion

NOVA Autonomous Inspection Robot is an evidence-aware active prototype: its implemented behavior and tests are real, while unbuilt hardware, deployment, and performance goals remain labeled. Completing the five integrations in dependency order will advance it from a learning artifact toward a credible portfolio component.

References

[1] Aaron Singh, "NOVA Autonomous Inspection Robot PROJECT.yaml," local portfolio repository.

[2] Aaron Singh, "NOVA Autonomous Inspection Robot: README, ANALYSIS, source, and test artifacts," local portfolio repository.

[3] Aaron Singh, "AI Infrastructure Portfolio Benchmark Standard," local portfolio repository.

TABLE II
AUDITED TEST MATRIX

Test artifact and case	Behavior being checked	Observed result
tests/test_inspection.py:test_brightest	Brightest.	Pass (local)
tests/test_inspection.py:test_no_defect	No defect.	Pass (local)
tests/test_inspection.py:test_path_ends_target	Path ends target.	Pass (local)
tests/test_inspection.py:test_bounds	Bounds.	Pass (local)

TABLE III
DECLARED STATUS VERSUS AUDITED EVIDENCE

Audit field	Finding	Evidence source
Declared status	Active Prototype	PROJECT.yaml
Evidence-backed status	Active local prototype; 4 tests passed	Source plus local unittest run
Accepted measured results	None recorded in measured_results	PROJECT.yaml
Mismatch / claim boundary	No camera, physical robot, or autonomous hardware inspection	PROJECT.yaml and ANALYSIS.md
Next proof required	Image dataset baseline; Kinematics simulator	Planned features

TABLE IV
PROJECT GOALS AND REQUIRED PROOF

ID	Goal	Completion evidence
G1	Image dataset baseline	Passing tests and a reviewed source artifact
G2	Kinematics simulator	Machine-readable result with reproduction metadata
G3	Collision and hardware safety	Reference-equivalence or domain-correctness report
G4	Establish an image-dataset baseline	Named profiler, deployment, or integration artifact
G5	Validate kinematics, collision, and stop behavior	Dashboard/report link preserving provenance and limitations