

SENSORGUARD: HARDWARE-BASED SENSOR INTEGRITY MONITOR FOR MOBILE ROBOTS

By

Chinmay Naveen

Sanjiv Kondin

Final Report for ECE 445, Senior Design, Fall 2026

TA: Alex Lee

18 September 2026

Abstract

Sensor Guard is a low-cost hardware module that can continuously cross-check the encoders of a robot's wheel against its IMU in order to detect sensor faults (dropped encoder pulses, IMU bias, frozen or corrupted readings) that individually look plausible but disagree with the robot's true motion. An STM32 controller does the computing for a time-windowed disagreement metric between the two independent motion estimates and maintains a real-time 0-100 health score per sensor, flagging a sensor and identifying which one is likely at fault once disagreement persists beyond normal noise. A built-in fault-injection subsystem lets us reproduce realistic, repeatable sensor failures on the bench. This proposal defines the problem, the high-level requirements, the design at subsystem-level and the safety and ethical considerations for this prototype.

Contents

1. Introduction.....	1
1.1 Section head.....	1
2 Design.....	2
2.1 [Component or Block].....	2
2.1.1 [Subcomponent or subblock].....	2
3. Design Verification.....	3
3.1 [Component or Block].....	3
3.1.1 [Subcomponent or subblock].....	3
4. Costs.....	4
4.1 Parts.....	4
4.2 Labor.....	4
5. Conclusion.....	5
5.1 Accomplishments.....	5
5.2 Uncertainties.....	5
5.3 Ethical considerations.....	5
5.4 Future work.....	5
References.....	6
Appendix A Requirement and Verification Table.....	7

1. Introduction

1.1 Problem

Small and mid-size robots depend almost entirely on wheel encoders and IMUs to estimate their own motion. Because of the fact that these two sensors typically fed a closed-loop motion controller directly, the controller has no real independent way of knowing whether the numbers that it is receiving are actually correct. A sensor does not have to fail to cause a problem. Instead, an encoder that begins missing pulses because of something like a wheel slip, or an IMU that develops a slowly growing bias, will end up producing values that look individually plausible even as they diverge from the robot's true motion.

This kind of silent and partial failure is actually very common in practice. A field-reliability study of a mobile robot fleet found that encoder and tires together accounted for almost 75% of all reported subsystem failures, with the encoder showing the lowest observed reliability of any measured

component. Meanwhile, a different study of wheeled robots were localization failures that were caused specifically by errors of odometry, with reported mean-time-between-failure on the order of tens of hours of operation. If we combine these results together, they underscore the failure mode that SensorGuard targets, which is one of the leading causes of navigation failure in fielded robots, not an edge case.

The consequence of an undetected sensor fault is not just a wasted run. This is because a robot that trusts a faulty encoder or a biased IMU can actually drift off its intended path, misjudge distance to obstacles or people, or even apply incorrect torque. All of these are safety concerns in environments where mobile robots share space with human workers and patients. As mobile robots are deployed more widely in our world, the cost of a controller silently acting on bad sensor data scaled from a minor inconvenience to a genuine public-safety hazard. There are a lot of existing approaches to this problem and they usually fall into two camps. 1 camp is full sensor-fusion state estimators, which can detect faults as a byproduct, but require significant computation and a complete localization stack, and simple range/bounds checks on individual sensors, which catch only gross readings and miss the subtle faults that we just described. There is a gap for a low-cost hardware layer that also sits between the raw sensors and the controller and answers one narrow practice question in real-time: can our robot still trust this particular sensor right now?

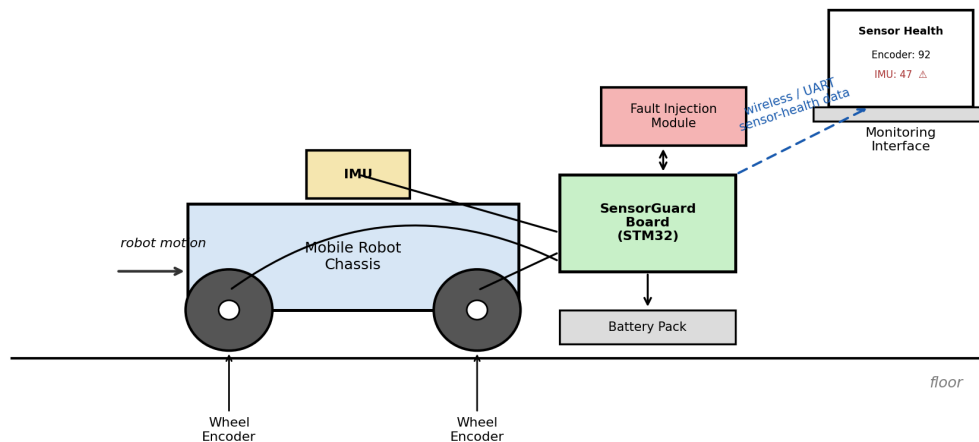
1.2 Solution

SensorGuard is a small low-cost hardware module that continuously cross-checks a robot's wheel encoders against its IMU and then reports a real-time confidence score for each sensor, without attempting to replace the robot's own localization or control stack. The core idea is that we have two independent sensing paths and they should agree about how the robot is moving. The microcontroller estimates rotational motion from different wheel-encoder counts from the IMU's gyroscope, and then computes a disagreement metrics between the two estimates of a short sliding time window, and then uses that disagreement to update a health score for each sensor. This health score will be from 0 to 100. When a sensor's score drops below a configurable threshold, SensorGuard flags it and reports which of the two sensing paths is the likely source of the fault to the robot's controller over UART/BLE, well before the disagreement would be large enough for a coarse threshold check to notice.

Concretely, the prototype uses an STM32 microcontroller in order to timestamp and decode signals from two encoders and read data from an IMU. A dedicated fault-injection subsystem lets people exercise SensorGuard against realistic, repeatable failure modes on the bench rather than waiting for a sensor to fail on its own. The resulting sensor-health values and fault flags are transmitted to a host monitoring interface and can also be consumed directly by the robot's existing motion controller as an additional input, giving the robot the option to reduce its reliance on a sensor if it can no longer trust it.

1.3 Visual Aid

SensorGuard Application Context

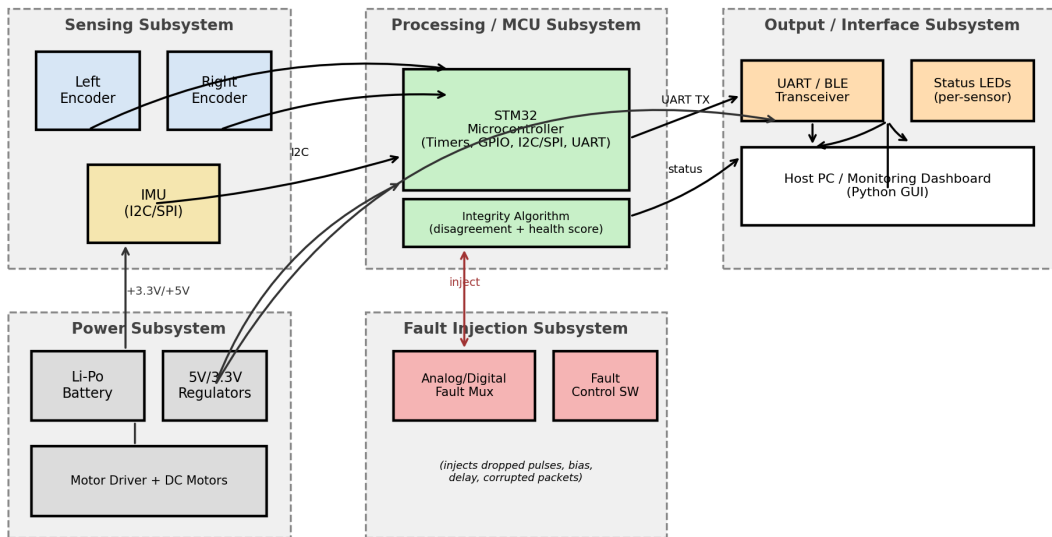


1.4 High Level Requirements

- SensorGuard must detect an injected sensor fault of at least $3^\circ/\text{s}$ of sustained angular-rate disagreement within 1.0 second of fault onset, with a false-positive rate below 5% during 10 minutes of normal driving.
- SensorGuard must independently sample both wheel encoders and the IMU at a minimum rate of 100 Hz per sensor so that the disagreement metric reflects the robot's actual motion rather than a stale estimate.
- SensorGuard must correctly identify which of the two sensing paths is responsible for an injected fault in at least 90% of single-sensor fault trials, and must report the corresponding health score to an external interface within 200 ms of the update.

2. Design

2.1 Block Diagram



Legend: solid black = data line | gray = power | red = fault-injection control

2.2 Subsystem Overview

Sensing Subsystem

The sensing subsystem consists of two incremental encoders and an IMU providing 3-axis acceleration and 3-axis angular rate. The encoders connect to the MCU subsystem's hardware timer peripherals for quadrature decoding, and the IMU connects over I2C or SPI. This subsystem provides the two physically independent estimates of robot motion that the rest of the system compares against each other.

Fault Injection Subsystem

This subsystem sits between the sensing subsystem and the processing subsystem and, under software control, can substitute a frozen signal for the true encoder or IMU signal on command. It is used only during bench testing and is disabled during normal operation, allowing the same physical test runs to be repeated under both healthy and faulty sensor conditions for evaluation.

Processing / MCU Subsystem

An STM32 microcontroller is the CPU of SensorGuard. It decodes the encoder signals, reads and timestamps IMU data, converts both into a common and normal estimate, computes the time-windowed disagreement metric between the two estimates, and updates each sensor's health score. This subsystem interfaces with every other subsystem since it receives raw sensor data from the sensing subsystems, draws regulated power from the power subsystem, and drives the subsystem with health scores and fault flags.

Power Subsystem

The power subsystem consists of a battery pack, buck regulators producing the 3.3 V and 5 V rails used by the MCU, IMU, and transceiver, and the motor driver stage that powers the robot's drive motors. It must supply regulated power to every other subsystem simultaneously, including during the current transients caused by motor operation.

Output / Interface Subsystem

The output subsystem consists of a UART transceiver, status LEDs, and a monitoring dashboard. It receives health scores and fault flags from the MCU and it is responsible both for giving a human observer an at-a-glance indication of sensor health and for making the same information available to the robot's own motion controller so that it can act on a degraded sensor.

2.3 Subsystem Requirements

Sensing Subsystem

This subsystem basically supplies the processing subsystem with two motion estimates: either if the path is too slow, too noisy, or if the disagreement metric becomes meaningless and SensorGuard cannot distinguish a real fault from normal sensor jitter.

Requirements:

- Each encoder must resolve at least 500 counts per wheel revolution.
- IMU must report angular rate at ≥ 100 Hz with gyroscope noise density ≤ 0.01 °/s/ $\sqrt{\text{Hz}}$.
- Encoder and IMU data must both be hardware-timestamped by the MCU to within 1 ms of acquisition.

Processing Subsystem

This subsystem is the one that the high-level requirements depend on the most directly. It has to run the disagreement and health-scoring algorithm fast and reliably enough in order to meet the 1 second detection and 200 ms reporting requirements without dropping any samples.

Requirements:

- MCU must sample both sensing paths and update the disagreement metric $E(t)$ at ≥ 50 Hz.
- MCU must maintain a persistent, exponentially-weighted health score per sensor, updated every cycle, retained across at least a 5 s rolling window.

Power Subsystem

Requirements

- Must supply a continuous $3.3 \text{ V} \pm 0.1 \text{ V}$ rail capable of sourcing at least 500 mA to the MCU, IMU, and transceiver.
- Must supply a continuous $5 \text{ V} \pm 0.25 \text{ V}$, 2 A rail to the motor driver stage, isolated so that motor current transients do not sag the 3.3 V logic rail by more than 50 mV.

Output Subsystem

Requirement:

- Must deliver health-score/fault-flag messages over UART at ≥ 115200 baud or BLE with ≤ 100 ms link latency.

2.4 Tolerance Analysis

The main aspect of the design that actually has the greatest risk to the success of our project is distinguishing a genuine sensor fault from something ordinary like sensor and timing noise using only the disagreement metric $E(t) = |M_{\text{encoder}}(t) - M_{\text{IMU}}(t)|$. If the natural, fault-free disagreement between the two sensing paths is comparable in magnitude to the smallest fault that we are required to detect, no threshold can separate the two cases and the detector will either miss real faults or false-trigger on healthy sensors.

We can estimate the expected fault-free disagreement from the individual sensor noise specifications. A typical low-cost MEMS gyroscope has an angular random walk on the order of $0.01^\circ/\text{s}/\sqrt{\text{Hz}}$, and that is sampled and averaged over a 1 Hz-equivalent bandwidth appropriate to a 1-second detection window, and this contributes roughly $0.01^\circ/\text{s}$ of noise directly to the IMU-side rotational estimate. The encoder-side rotational-rate estimate that is derived from differential wheel counts has an angular error of approximately: $(2\pi/N) \times (r/L)$. In this equation, N is the encoder resolution in counts per revolution, r is the wheel radius, and L is the wheel separation. For $N = 500$ counts/rev, $r = 0.03$ m, and $L = 0.15$ m, and this gives an angular resolution per encoder tick of approximately 0.05% of robot heading change, which at a nominal update rate of 50Hz corresponds to an equivalent angular-rate noise of roughly $2.5^\circ/\text{s}$ per tick edge, which drops to well under $0.5^\circ/\text{s}$ once the velocity estimate is filtered over a rolling window of 10-20 samples.

If we combine the IMU and encoder noise contributions, we get an expected 1σ fault-free disagreement on the order of $0.4\text{-}0.6^\circ/\text{s}$, which is roughly a factor of 5-7 below the magnitude that the system actually must detect. This margin is the justification for requiring $E(t)$ to persist above threshold for a multi-sample window rather than triggering on a single sample. The main residual risk is that the real-world noise may exceed our estimate, and the early-benchmark characterization work is specifically intended to measure the true fault-free disagreement distribution and re-tune the detection threshold and window length before fault-injection testing starts.

3. Ethics, Safety, and Societal Impact

SensorGuard is intended to make mobile robots safer by surfacing sensor problems that would otherwise go unnoticed, but the project itself raises ethical and safety considerations that the team must actively manage rather than assume away. The IEEE Code of Ethics calls on engineers to hold paramount the welfare of the public and to disclose factors that might endanger the public, and the ACM Code of Ethics similarly talks about avoiding harm and being honest about the limitations of a system's what SensorGuard can and cannot do.

The most significant ethical risk is overstatement of reliability: because SensorGuard reports a continuous confidence score rather than a binary pass/fail result, there is a temptation to imply that a

high score guarantees a sensor is trustworthy. In reality, the detection thresholds developed are tuned against a specific platform and we will document these boundaries explicitly, avoid marketing language suggesting general-purpose certification of sensor safety, and clearly state in the final report and demonstration which fault classes were and were not validated. This mirrors the ACM Code's emphasis on being transparent about system limitations and avoiding false claims about a system's fitness for a particular use and should abide to standards overall.

During development, we will observe standard electrical and mechanical safety practices. The prototype operates from a low-voltage power budget, so the primary hazards are Li-Po battery mishandling and will test fault-injection scenarios that could cause erratic robot motion only in a bounded, low-speed test area with the ability to immediately cut motor power, consistent with campus lab safety policy for mobile robotics projects.

4. Costs

4.1 Parts

Part	Manufacturer	Qty	Cost (\$)
STM32 Nucleo-F446RE dev board	STMicroelectronics	1	13.50
MPU-6050 6-axis IMU breakout	InvenSense	1	6.95
600 P/R incremental rotary encoder	E38S series	2	9.00 ea.
Analog/digital multiplexer (fault injection)	Texas Instruments (CD74HC4051)	2	1.20 ea.
Dual H-bridge motor driver	TB6612FNG	1	4.95
2S Li-Po battery pack (1300 mAh)	Turnigy	1	9.50
Buck regulator boards (5 V, 3.3 V)	Pololu	2	5.95 ea.
Differential-drive chassis + DC motors + wheels	generic robotics kit	1	28.00
HC-05 UART/BLE transceiver module	generic	1	7.50
Status LEDs, connectors, protoboard, misc. hardware	—	—	10.00
Total			\$106.10

References

- [1] P. H. Tsarouhas and G. K. Furlas, "Mission reliability estimation of mobile robot system," International Journal of System Assurance Engineering and Management, vol. 7, no. 2, 2016.
- [2] A. Korodi and T. L. Dragomir, "Correcting odometry errors for mobile robots using image processing," in Proc. IMECS 2010, Hong Kong, 2010, pp. 1040–1045.
- [3] P. Sundvall and P. Jensfelt, "Fault detection for mobile robots using redundant positioning systems," in Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), 2006.

[4] M. Hashimoto, H. Kawashima, and F. Oba, "Sensor fault detection and diagnosis in dead reckoning system of mobile robot: Interacting multiple-model approach," *Journal of Robotics and Mechatronics*, vol. 14, no. 4, pp. 342–348, 2002.

[5] IEEE Code of Ethics, IEEE, 2020.

[6] ACM Code of Ethics and Professional Conduct, Association for Computing Machinery, 2018.