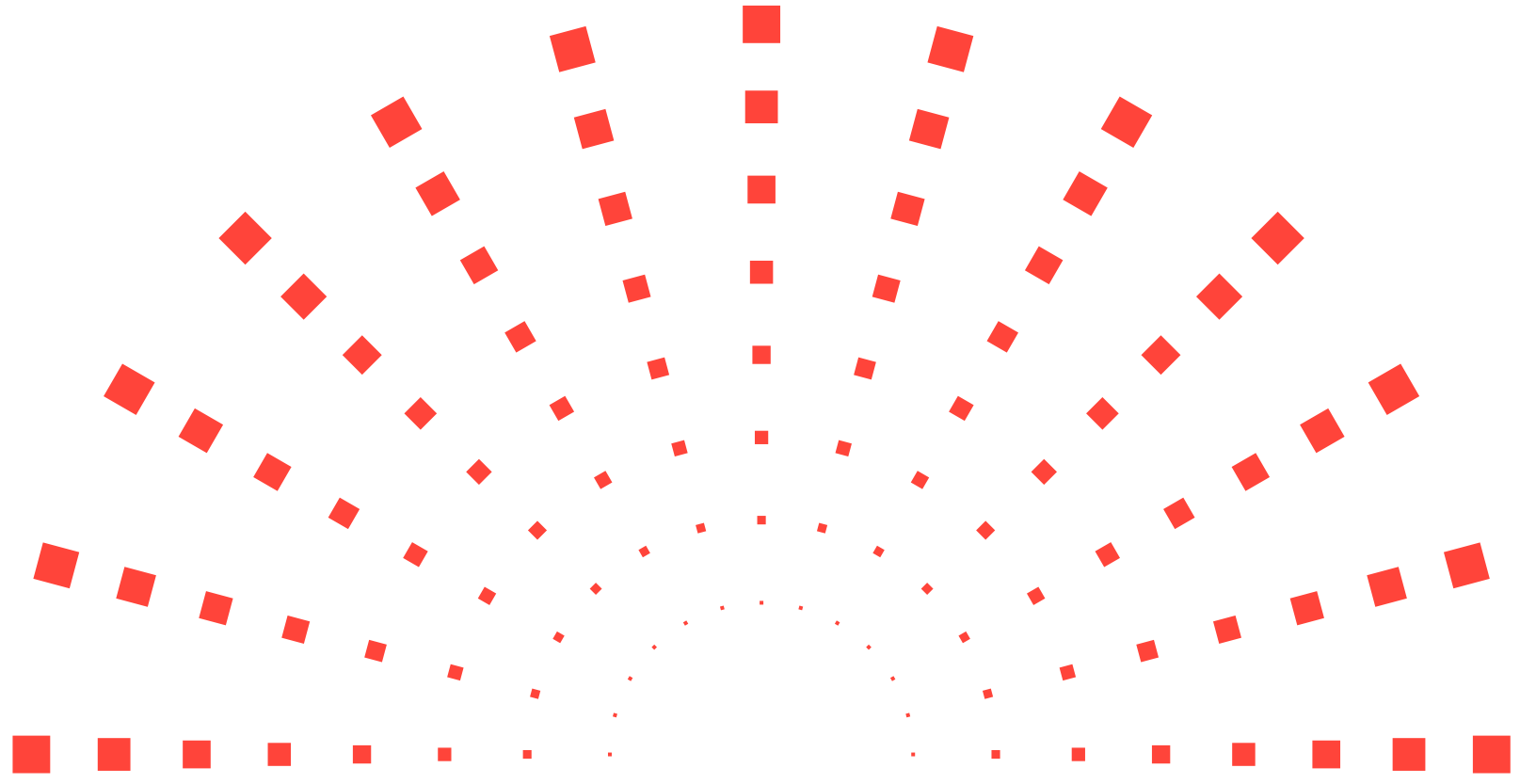
QNX | CYBERWORKS ROBOTICS

White Paper

A QNX-Based Active Safety System for Mission-Critical Autonomous Mobility

Safety is not something added after autonomy works—it's something that must be built into the system architecture from the outset



Overview

Early generations of mobile robots were deployed into environments that were mobile in form, but static in assumption. Warehouses, factories, and logistics centers were carefully structured to accommodate robotic motion: aisles were fixed, traffic patterns were prescribed, and human access to robot operating zones was tightly controlled. Even though the robots themselves moved, their environments were effectively segregated. Safety was achieved through separation, physical barriers, restricted zones, and procedural controls that limited when and where humans and robots could coexist.

In this model, safety architecture was straightforward. Mobile robots navigated predefined spaces and executed well-understood behaviors, while safety was enforced externally through simple, independent mechanisms. Emergency stops, safety PLCs, and microcontrollers provided a hard boundary: if a human entered a protected zone or a gate opened, the robots shut down. Motion was removed, and with it, the immediate hazard. The environment was predictable enough that this approach worked, and the operational impact of stopping was acceptable. The assumptions that underpinned traditional industrial safety no longer hold.

Today's mobile robots are being pressed into environments that are no longer segregated, no longer predictable, and no longer static. Robots now operate in the same unstructured spaces as humans—sharing aisles, corridors, work areas, and decision-making space with people whose behavior cannot be scripted or constrained. Paths are dynamic. Objects are transient. Human presence is continuous rather than exceptional. In these environments, separation is not the norm; coexistence is.

The safety problem changes fundamentally when separation disappears. A gate opening is no longer an exceptional event, it's nominal. If the legacy safety response is applied unchanged, the result is a robot that stops constantly. While stopping may be safe in isolation, it is operationally fragile. Frequent emergency stops reduce availability, create traffic stand-offs, and introduce new risks as humans and robots negotiate around immobile machines. The robot becomes safe, but immobile; predictable, but unproductive.

This exposes a deeper architectural mismatch. Traditional mobile robot safety treated motion as something to be revoked under uncertainty. That mindset assumed that safety could live outside the robot's intelligence, enforced by independent controllers whose primary function was to remove authority. In shared human environments, that assumption breaks down. Safety should no longer be a binary condition triggered by boundary violations. It should be continuous, contextual, and deeply integrated with how the robot perceives, plans, executes, and monitors motion.

As mobile robots transition from segregated operation to true human-shared environments, the definition of "safe behavior" must evolve. Stopping remains a valid fallback, but it can no longer be the default response to uncertainty. Instead, robots must be able to slow, yield, reroute, maintain separation, and move around people and obstacles in a way that is both safe and efficient. This demands architectures that treat safety not as an external override, but as a priority of motion itself—enforced at runtime, under uncertainty, and at the same temporal scale as navigation and control.

The environments have changed. The expectations have changed. The architectures and safety mentalities that underpin mobile robotics must change with them.

System Design Considerations

Designing mobile robots for shared, unstructured environments requires safety systems to be integrated rather than as a discrete subsystem. It becomes a property of the entire system, emerging from how sensing, computation, software execution, and actuation are composed and governed end to end. Designing a safe mobile robot in this context requires a shift in mindset: safety is not something added after autonomy works—it's something that must be built into the system architecture from the outset.

At the foundation of this architecture is the execution environment itself. A safety-certified IEC 61508 SIL 3 real-time operating system provides far more than basic task scheduling. It establishes the deterministic behavior, fault containment, and freedom from interference that higher-level safety logic depends on. In a shared environment, safety-critical functions cannot compete unpredictably with complex autonomy workloads for CPU time, memory, or I/O. The operating system must allow safety functions to run with bounded latency, explicit priority, and, where necessary, dedicated processor cores. This execution foundation is what allows safety decisions to be enforced reliably rather than best-effort.

Sensors represent the next critical layer in the safety argument. In traditional designs, sensors primarily served navigation performance. In human-shared environments, they become safety-relevant inputs. A safe mobile robot typically combines safety-certified sensors with redundant and complementary sensing modalities to reduce common-cause failures and ensure coverage in all directions of motion. Just as importantly, sensors must be actively managed rather than passively consumed. Their health, freshness, and confidence must be observable by the system, and degradation must be detectable. Safety is not achieved by assuming perception is correct, but by knowing when perception is no longer sufficient to support safe motion.

Perception and navigation sit at the boundary between raw sensing and physical action, and in modern mobile robots they directly influence safety, even when they are not themselves fully safety-certified. These components are responsible for transforming sensor data into an understanding of free space, obstacles, human proximity, and environmental dynamics, and then turning that understanding into motion decisions. From a system-design perspective, the key requirement is that perception and navigation produce safety-relevant abstractions rather than opaque decisions. Free-space estimates, clearance margins, feasible trajectories, and confidence bounds must be explicit and consumable by safety logic. Safety does not rely on perception being perfect; it relies on perception being honest about uncertainty.

This leads to a fundamental departure from legacy safety models. In shared environments, safety cannot be binary. Intervening only when a hard limit is violated (e.g. by stopping or removing power) preserves safety but at the cost of frequent disruptions and loss of productivity. Instead, safety must be enforced continuously. Speed must be modulated based on available stopping distance and sensor coverage. Trajectories must be constrained to verified-safe corridors. Motion must be biased away from humans dynamically, and behavior must failover gracefully as uncertainty increases. Stopping remains a valid fallback, but it becomes the last resort rather than the default response.

Achieving this requires tight coupling between perception, navigation, control, and safety supervision. Safety logic must operate at the same temporal scale as motion planning and control, shaping behavior in real time rather than reacting after the fact. This does not mean that all autonomy must be safety-certified, but it does mean that safety authority must be clearly defined and retained by components that are deterministic, bounded, and verifiable. Regardless of how advanced the autonomy becomes, there must be an unambiguous answer to who ultimately owns the right to command motion.

Finally, safe mobile robot architectures must be designed with evolution in mind. The environments robots operate in will continue to change, and autonomy will become more

capable and more complex over time. A robust system design allows perception and navigation algorithms to evolve without requiring a complete re-justification of the safety case. By keeping safety authority stable, enforcing clear interfaces, and constraining complex behaviors rather than trusting them, the system can incorporate more advanced AI while maintaining a consistent and credible safety story.

This architectural perspective sets the stage for a deeper exploration of the software stack. With the execution foundation, sensing strategy, and safety philosophy established, we can now examine how perception, navigation, and control software are structured to support continuous, optimized, and dependable safety in real-world mobile robotic systems.

The Challenge

Despite rapid advances in autonomy, most mobile robotic systems today were not designed with safety as a holistic, system-level property. In practice, many robots are still built on general-purpose, non-safety-certified operating systems, with soft real-time behavior layered on top of complex autonomy stacks. Control, perception, and navigation are often scheduled opportunistically rather than deterministically, and safety logic is added as an external mechanism rather than an intrinsic capability. This approach may work in controlled conditions, but in real deployments it exposes a fundamental tension: when timing, resource contention, or perception uncertainty increases, the system's only reliable response is to stop.

Historically, safety in robotics has been implemented through emergency stops and hard overrides, and for good reason. Stopping is simple, verifiable, and effective at preventing immediate harm. However, in human-shared environments, this model increasingly works against the very goals developers are trying to achieve. Robots that stop frequently are safe, but they are not dependable. They block aisles, create operational stand-offs with humans, and introduce unpredictable delays that ripple through workflows. Over time, these interruptions erode its value. The robot is no longer perceived as an asset that improves efficiency, but as a bottleneck that must be worked around.

As mobile robots are introduced into increasingly dynamic and human-shared spaces, the limits of traditional safety approaches are becoming more visible. Environments where stopping was once acceptable are giving way to workflows that demand continuity, predictability, and smooth interaction between people and machines. In this context, availability becomes inseparable from safety: a robot must not only avoid harm, but do so in a way that allows it to remain a dependable participant in the operation. This shift is creating a need for new safety architectures, ones that move beyond binary interventions and instead support continuous, constrained motion. Meeting this need does not reflect a failure of past designs, but rather the natural evolution of robotics into domains that demand a different balance between safety, performance, and reliability.

The Solution: Safe-By-Design Architecture

The difficulty most teams encounter when deploying mobile robots into human-shared environments is not rooted in autonomy capability, but in system design mismatches. Navigation and perception stacks have become increasingly sophisticated, yet they are still commonly executed on non-safety-certified, best-effort software platforms with limited guarantees around timing, fault containment, and recovery. Safety, meanwhile, is often implemented as an external mechanism—an emergency stop chain, a safety PLC, or a watchdog—that operates outside the autonomy stack and intervenes only when predefined thresholds are crossed.

This architectural separation creates a fragile system. Transient failures such as sensor dropouts, process crashes, timing inconsistencies, or short-lived localization instability are treated the same as critical hazards: the robot stops. From a narrow safety perspective this is acceptable. From an operational perspective, it is not. Availability collapses, recovery requires human intervention, and robots become unpredictable participants in shared workflows.

Cyberworks Robotics' OMNISuite, powered by QNX, was designed to address this gap by treating autonomy, safety, and availability as interlocking software concerns, rather than independent subsystems. The stack assumes from the outset that faults will occur—not just catastrophic failures, but transient, recoverable ones—and that the system must respond deterministically, proportionally, and traceably.

At the foundation of OMNISuite is a real-time execution environment built on QNX, selected deliberately for its microkernel architecture and its evolution toward high-throughput, multi-core scalability. This choice reflects a core design principle: safety and security must be inherent to the system architecture, without constraining performance or limiting system growth.

The QNX microkernel minimizes the trusted computing base by confining only essential services to the kernel, while all other services execute in isolated user space. This design enables strong fault containment and controlled failure modes across perception, navigation, planning, application logic, and supervisory functions alike. The architecture allows each to be executed with isolation and guarantees appropriate to its role in system behavior.

Crucially, this isolation does not constrain performance or throughput. QNX has evolved its kernel and message-passing architecture to scale efficiently across high core-count and heterogeneous compute platforms. Fine-grained scheduling and scalable IPC allow compute-intensive perception and planning pipelines to fully utilize available processing resources, while functions requiring stricter real-time guarantees execute with bounded latency and predictable behavior.

By supporting the QNX Operating System (QOS) with a unified safety framework, OMNISuite avoids treating safety as a bolt-on or supervisory afterthought. Instead, it provides an execution foundation where autonomy, perception, and decision-making are all designed, scheduled, and isolated as safety-relevant system elements—allowing capability to scale without eroding determinism, containment, or trust.

Within this environment, OMNISafe, the OMNISuite safety supervisor, functions as an active safety supervisor rather than a passive interlock. It operates independently of the navigation stack, continuously monitoring the integrity of the robot's hardware, software, and motion behavior. This independence is critical: safety enforcement must remain functional even when autonomy software is degraded or unavailable.

From a sensing perspective, OMNISafe treats sensors as safety-relevant system components rather than raw data sources. Cameras, LiDAR, IMUs, wheel encoders, and other inputs are continuously validated for presence, minimum data rate, and temporal consistency. Timestamp validation and cross-sensor correlation are used to detect subtle failure modes—such as stale data or silent degradation—that would otherwise propagate unnoticed into navigation decisions. Safety, in this context, is about detecting when the system's view of the world is no longer reliable enough to support the current mode of motion.

At the software layer, OMNISafe supervises both ROS nodes and native QNX processes. It detects crashes, hangs, and abnormal execution patterns, and applies deterministic enforcement through watchdog mechanisms and controlled recovery actions. Unlike traditional safety approaches that respond to any fault with an immediate shutdown, OMNISafe distinguishes between transient failures and persistent or high-risk conditions. Short-duration issues such as camera disconnects, data dropouts, temporary navigation instability, or brief power fluctuations are detected, isolated, and recovered from autonomously where safe to do so. This capability is essential for maintaining uptime in real-world environments where such disturbances are unavoidable.

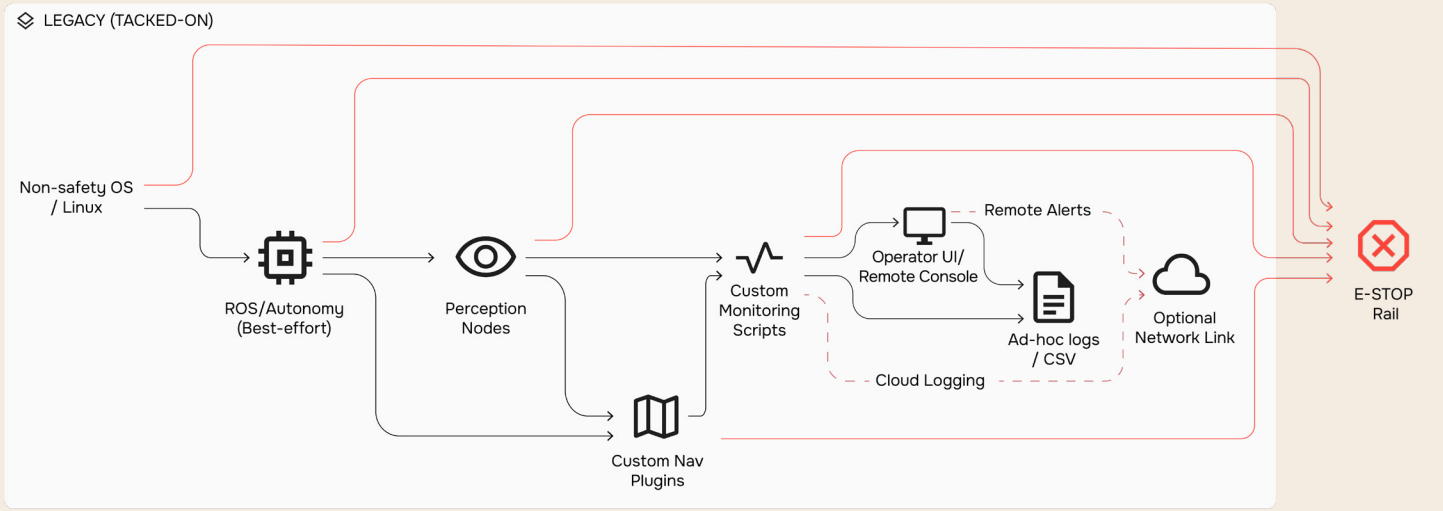
Navigation and motion behavior are also treated as safety-relevant signals. OMNISafe monitors localization consistency, steering behavior, and acceleration profiles to detect anomalies that may indicate drift, control instability, or unexpected deviation from planned motion. Rather than assuming that navigation outputs are correct, the system continuously evaluates whether observed motion remains consistent with expected behavior and physical constraints.

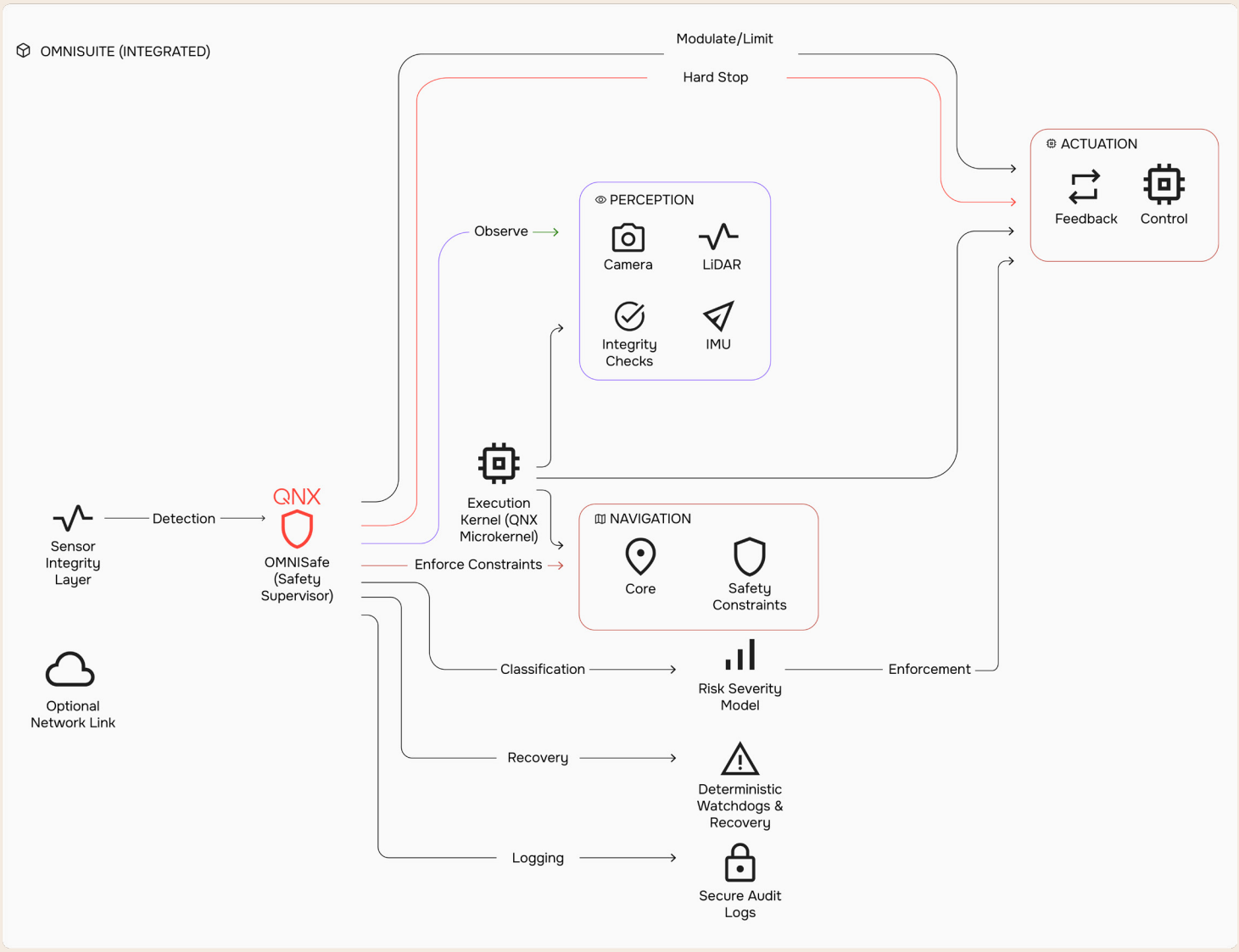
Central to the design is its structured failure severity and response model. All detected conditions are classified deterministically into predefined severity levels—high, medium, low, or ignore—each mapped to a specific set of enforcement actions. High-risk conditions result in immediate containment, including robot halt, user notification, secure logging, and remote alerting. Medium-risk conditions pause operation and require acknowledgment before proceeding. Lower-risk issues are logged for diagnostics without interrupting operation. This explicit mapping replaces ad hoc fault handling with predictable, certifiable behavior, making system response both understandable to operators and defensible to regulators.

Equally important is traceability. OMNISafe maintains secure, audit-ready logs that support post-incident reconstruction, certification workflows, and regulatory review. Safety is not only enforced at runtime; it is documented continuously, providing the evidence required to demonstrate that the system behaves as intended under fault conditions.

Taken as a whole, OMNISuite represents a shift away from stop-based safety toward managed, continuous safety enforcement. Emergency stops remain part of the system, but they are no longer the primary tool for handling uncertainty. Instead, safety is expressed through supervision, classification, constraint, and recovery—allowing robots to remain available, predictable, and trustworthy in the environments they are increasingly expected to operate in.

This system-level approach is what enables mobile robots —and eventually humanoids and other highly dynamic platforms—to move beyond brittle autonomy and into truly shared spaces. In the next section, we will examine how these principles are realized within the software stack itself, from perception abstractions to navigation constraints and runtime safety authority.


LEGACY (TACKED-ON)
Non-safety OS / Linux
ROS/Autonomy (Best-effort)
Perception Nodes
Custom Monitoring Scripts
Custom Nav Plugins
Operator UI/ Remote Console
Remote Alerts
Ad-hoc logs / CSV
Cloud Logging
Optional Network Link
E-STOP Rail


OMNISUITE (INTEGRATED)
Modulate/Limit
Hard Stop
ACTUATION
Feedback
Control
PERCEPTION
Observe
Camera
LiDAR
Integrity Checks
IMU
Execution Kernel (QNX Microkernel)
NAVIGATION
Core
Safety Constraints
Sensor Integrity Layer
Detection
QNX
OMNISafe (Safety Supervisor)
Enforce Constraints
Optional Network Link
Classification
Risk Severity Model
Enforcement
Recovery
Deterministic Watchdogs & Recovery
Logging
Secure Audit Logs

Conclusion

As autonomous mobility systems become integral to industrial, commercial, and public environments, safety must evolve from an afterthought to a foundational design principle. Cyberworks Robotics' OMNISuite platform, running on QNX, delivers the deterministic, certifiable, and real-time safety infrastructure required for the next generation of autonomous systems.

By integrating OMNISuite, OEMs and developers can accelerate time-to-market, reduce risk, and deploy autonomous mobility solutions that meet the highest standards of safety and reliability.

About QNX

QNX, a division of BlackBerry Limited, enhances the human experience and amplifies technology-driven industries, providing a trusted foundation for software-defined businesses to thrive. The business leads the way in delivering safe and secure operating systems, hypervisors, middleware, solutions, and development tools, along with support and services delivered by trusted embedded software experts. QNX® technology has been deployed in the world's most critical embedded systems, including more than 275 million vehicles on the road today. QNX® software is trusted across industries including automotive, medical devices, industrial controls, robotics, commercial vehicles, rail, and aerospace and defense. Founded in 1980, QNX is headquartered in Ottawa, Canada.

Learn more at qnx.com →