



Journal of Interdisciplinary Science & Technology

Design and Implementation of a Cloud-Edge Collaborative Mapping and Control System for Virtual Mobile Robots: A Review

Prof. Varsha Ohol¹, Vivek Gunvant Patil¹, Tejas Rajendra Badgujar¹,
Sharvil Vijayanand Lahute¹, Harshal Bharat Bagul¹

¹Department of Computer Engineering, Sandip Institute of Technology and Research Centre, Nashik, Maharashtra, India
varsha.ohol@sitrc.org, vivekpatil205@gmail.com, tejasbadgujarbsl@gmail.com, sharvillahute1991@gmail.com,
harshalbagul0987@gmail.com

Corresponding author- Prof. Varsha Ohol

DOI: 10.5281/zenodo.23539241

Abstract: The increasing use of autonomous mobile robots in warehouses, healthcare, manufacturing, logistics, agriculture, surveillance, and disaster response has created a need for systems that combine responsive local control with scalable remote services. Conventional robots execute mapping, localization, path planning, obstacle avoidance, monitoring, and data management on limited onboard hardware. Cloud-only designs provide elastic computation and centralized visibility but expose time-critical control to network delay and disconnection. This review studies robot middleware, simulation, simultaneous localization and mapping, autonomous navigation, cloud robotics, edge computing, and cloud-native deployment. Based on the synthesis, a Cloud-Edge Collaborative Mapping and Control System for Virtual Mobile Robots is proposed. ROS 2 Jazzy and Gazebo Harmonic form the virtual-robot and edge layers; SLAM, localization, and Nav2 retain the low-latency control loop; FastAPI and WebSockets provide cloud communication; Redis and PostgreSQL manage transient and durable data; React.js provides monitoring and control; and Docker with Kubernetes/Minikube supports reproducible deployment. The proposed architecture is simulation-first, cost-effective, modular, and suitable for academic validation before physical-robot integration. The paper defines its component boundaries, data flow, evaluation methodology, risks, and future extensions without presenting uncollected experimental results.

Keywords: Cloud robotics, edge computing, ROS 2, Gazebo, SLAM, Nav2, autonomous navigation, FastAPI, WebSocket, Docker, Kubernetes.

I INTRODUCTION

Autonomous mobile robots must transform continuous observations into safe action. LiDAR, cameras, inertial sensors, wheel odometry, and transform frames support environment mapping and localization, while global and local planners convert goals into collision-free motion. ROS established a reusable node-and-message model for robotic software [1], and ROS 2 extends that model with DDS communication, quality-of-service controls, actions, and distributed deployment [2]. Gazebo provides repeatable physics simulation and configurable virtual sensors, enabling the complete software chain to be exercised before physical hardware is available [3]. Placing every function onboard avoids network dependence but increases CPU, memory, storage, thermal, and battery requirements. Conversely, moving navigation and control into a remote cloud can make safety depend on latency, jitter, packet loss, or service availability. Edge computing addresses this

tension by retaining interactive computation near its data and actuator while cloud services handle suitable centralized work [9], [10]. Cloud robotics additionally enables shared maps, remote monitoring, data-intensive analysis, and fleet-level coordination [11], [12].

The project reviewed in this paper applies these principles to a virtual mobile robot. It keeps sensor handling, SLAM, localization, obstacle response, planning, and velocity control at the ROS 2 edge. FastAPI cloud services manage robot identity, telemetry, maps, tasks, logs, and health. WebSockets deliver real-time state to a React.js dashboard, Redis maintains rapidly changing state, PostgreSQL stores durable records, and containers are orchestrated through Kubernetes/Minikube. The result is a cost-effective research platform that can be migrated to physical robots later.

The main contributions of this paper are:

- A structured review of robotics middleware, SLAM, navigation, cloud robotics, edge computing, web communication, and container orchestration.
- A system architecture that separates safety-critical edge functions from scalable cloud services and operator-facing visualization.
- An implementation methodology using ROS 2 Jazzy, Gazebo Harmonic, Nav2, FastAPI, React.js, Redis, PostgreSQL, Docker, Kubernetes, and Minikube.
- A measurable evaluation plan for latency, mapping quality, navigation success, resources, reliability, and failure recovery.

II. REVIEW METHOD AND RELATED WORK

The literature was organized into five technical themes: robot middleware and simulation; mapping and localization; autonomous navigation; cloud and edge robotics; and cloud-native service infrastructure. The review prioritizes seminal peer-reviewed papers, official technical documentation for the selected stack, and standards that define communication behavior. The goal is an architecture-oriented synthesis rather than a statistical meta-analysis.

A. Robot Middleware and Simulation

ROS introduced reusable packages, message passing, tools, and common abstractions for robotics research [1]. ROS 2 preserves this modularity while improving distributed communication, security support, and control over message reliability and durability [2]. These features are important when a robot bridge exchanges state with cloud services but the local navigation graph must continue operating during partial connectivity.

Gazebo Harmonic simulates rigid-body physics, worlds, sensors, and robot actuation [3]. Simulation reduces hardware cost, improves repeatability, and permits controlled failure testing. It does not reproduce every source of wheel slip, calibration drift, sensor noise, radio interference, or actuator wear; therefore, successful simulation is a prerequisite for physical deployment rather than proof of field readiness.

B. Mapping and Localization

SLAM jointly estimates a robot trajectory and an environment map from controls and observations. Durrant-Whyte and Bailey formalized the probabilistic foundations and consistency challenges of SLAM [5]. Grid-based approaches such as GMapping combine Rao-Blackwellized particle filters with improved proposals and adaptive resampling to produce occupancy maps from range observations [6]. After a map is saved, probabilistic localization methods such as Monte Carlo localization estimate robot pose within the map [7].

These algorithms are sensitive to frame consistency, timestamp quality, odometry error, sensor configuration, and map resolution. The proposed system therefore retains transforms, sensor preprocessing, SLAM, and localization at the edge. The cloud receives selected map artifacts and telemetry but is not inserted into the estimator's real-time update loop.

C. Autonomous Navigation

Nav2 provides planning, control, layered costmaps, behavior trees, recovery actions, and goal execution for ROS 2 robots [4]. Its design supports modular selection of global planners, local controllers, filters, and recovery behavior. Macenski et al. describe Nav2 as a production-grade navigation framework and emphasize its extensibility and lifecycle-managed components [8]. The system uses Nav2 at the edge so that obstacle avoidance and motion remain available when the dashboard or cloud backend is unavailable.

D. Cloud and Edge Robotics

Satyanarayanan identifies latency, bandwidth, privacy, and context as central motivations for edge computing [9], while Shi et al. describe the broader vision and challenges of moving computation toward the network edge [10]. Cloud robotics surveys show that shared infrastructure can support computation, data, knowledge, and coordination beyond the capability of an individual robot [11]. The cloud-robotics model proposed by Goldberg and Kehoe similarly frames the cloud as a source of shared data, computation, and collective learning [12].

The literature suggests a placement rule rather than a cloud-first rule: functions with hard or safety-related deadlines must remain near the robot; functions benefiting from aggregation, persistence, remote access, or elastic capacity may be placed in cloud services. A reliable system must also define behavior during delay, duplication, reordering, stale state, and disconnection.

E. Web and Cloud Infrastructure

FastAPI offers typed request validation, asynchronous Python endpoints, and WebSocket support [13]. React provides a component model for an interactive browser dashboard [14], while RFC 6455 defines full-duplex WebSocket communication over a single connection [15]. PostgreSQL is selected for durable relational state [16], and Redis supports transient state, caching, and messaging [17]. Docker packages services with explicit dependencies [18]; Kubernetes manages deployment, discovery, restart, and scaling [19]; and Minikube provides a local Kubernetes environment suitable for academic development [20].

III. RESEARCH GAP AND PROBLEM DEFINITION

Many academic robot projects stop at simulation and local visualization, whereas cloud prototypes may treat the robot as a generic IoT source. Industrial cloud-robotics platforms can be hardware-dependent, proprietary, or too costly for small academic teams. A practical gap therefore exists for a unified, simulation-based platform that integrates realistic robot navigation, clear cloud-edge boundaries, real-time monitoring, durable data, and cloud-native deployment using accessible open technologies.

The problem is to design a modular mapping and control system that preserves the responsiveness and safety of local navigation while enabling centralized tasks, telemetry, maps, logs, and visualization. The design must avoid making wheel control or obstacle response dependent on a cloud round trip. It must also support future migration from a virtual robot to physical sensors and actuators without replacing the cloud contract.

For a robotic function f_i with computation time c_i , network delay n_i , serialization and queuing overhead q_i , and deadline d_i , remote execution is eligible only if:

$$c_i + n_i + q_i < d_i. \quad (1)$$

Safety-critical functions use the stronger rule that they remain in the edge set regardless of average cloud performance. Consequently, emergency handling, sensor transforms, localization, obstacle response, planning, and velocity control are placed at the edge. Task history, dashboards, long-lived maps, and system-wide monitoring are placed in the cloud.

IV. PROPOSED SYSTEM ARCHITECTURE

The proposed system is organized as a closed cloud-edge loop with four principal layers. The presentation layer displays state and accepts authorized high-level goals. The cloud layer validates requests, manages tasks, stores records, and streams telemetry. The edge layer translates permitted tasks into ROS 2 actions and performs mapping and navigation. The virtual-robot layer supplies sensors and actuators through Gazebo. Figure 1 shows the architecture and the separation between telemetry/task communication and the low-latency control loop.

The Figure 1 which explains cloud edge collaborative system is depicted in the next section of the manuscript.

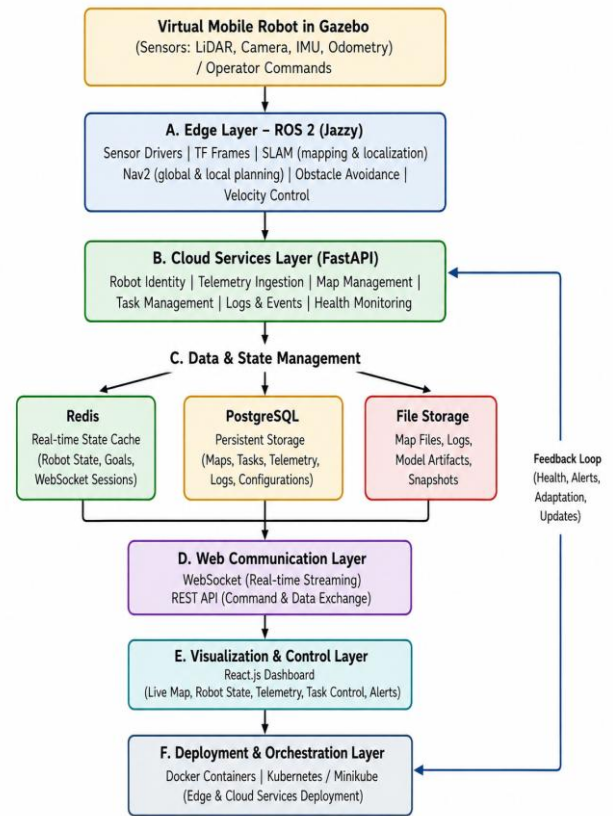


Figure 1: Cloud-edge collaborative system architecture for the virtual mobile robot.

A. Virtual Robot and Sensor Layer

The Gazebo robot uses a differential-drive base and virtual LiDAR, camera, IMU, and odometry sources. The robot description defines links, joints, collision geometry, inertial properties, sensors, and transforms. Representative worlds contain walls, corridors, and obstacles that exercise mapping and navigation. Gazebo publishes observations through ROS 2 interfaces and applies velocity commands to the virtual drive system.

B. ROS 2 Edge Layer

The edge hosts robot drivers, transform publication, sensor preprocessing, SLAM, localization, Nav2, and the cloud bridge. SLAM generates an occupancy map during exploration. Localization estimates pose in a saved map. Nav2 accepts a high-level goal, plans a global route, maintains local costmaps, generates control commands, and invokes recovery behavior when progress is blocked. The bridge exposes only an allowlist of high-level tasks; cloud text or data cannot directly become actuator commands.

C. Cloud Service and Data Layer

FastAPI manages robot registration, connection state, telemetry ingestion, task creation, map metadata, operational logs, and health endpoints. REST is used for bounded request-response

operations. WebSockets carry changing pose, navigation state, alerts, and task progress. Each message contains a robot identifier, timestamp, schema version, sequence number, and payload. These fields permit the receiver to detect stale, reordered, duplicated, or incompatible data.

Redis stores current robot state, cache entries, and short-lived messaging information. PostgreSQL stores robots, users, tasks, maps, events, and task-state history. Durable task identifiers make retries idempotent: a repeated request returns the existing task rather than creating duplicate navigation commands.

D. Dashboard and Deployment Layer

The React.js dashboard shows connection health, robot pose, velocity, active task, navigation status, occupancy map, planned route, goal, alerts, and logs. Operator actions submit high-level goals or cancellation requests through authenticated endpoints. The dashboard is not a safety controller and cannot bypass edge validation.

Docker images package the API, dashboard, and supporting services with pinned dependencies. Kubernetes/Minikube supplies declarative deployments, service discovery, configuration, health checks, resource requests, and restart policies. Containerization must be tested with the chosen DDS and network configuration because ROS 2 discovery behavior can vary across host and container boundaries.

V. DETAILED METHODOLOGY

A. Development Sequence

Implementation proceeds incrementally. First, the robot model, transforms, sensors, and teleoperation are validated in Gazebo. Second, SLAM is configured and the map is saved. Third, localization and Nav2 are tested against fixed start and goal poses. Fourth, the edge bridge publishes selected telemetry and accepts a restricted task schema. Fifth, FastAPI, Redis, and PostgreSQL are integrated. Sixth, the React dashboard visualizes data and submits authorized tasks. Finally, Docker images and Kubernetes manifests are tested in Minikube.

B. Data and Task Flow

During operation, Gazebo produces sensor and odometry messages. Edge nodes maintain transforms and estimate the robot state. Nav2 executes a goal locally and reports lifecycle updates. The bridge converts selected state into versioned cloud messages. FastAPI validates and persists the data, updates Redis, records durable events in PostgreSQL, and broadcasts changes to subscribed dashboards. A dashboard goal follows the reverse route: authenticated request, schema validation, robot and task eligibility checks, cloud task record, edge delivery, ROS 2 action acceptance, and status feedback. Figure 2 illustrates the closed-loop agent–environment interaction and task scheduling flow across edge and cloud layers.

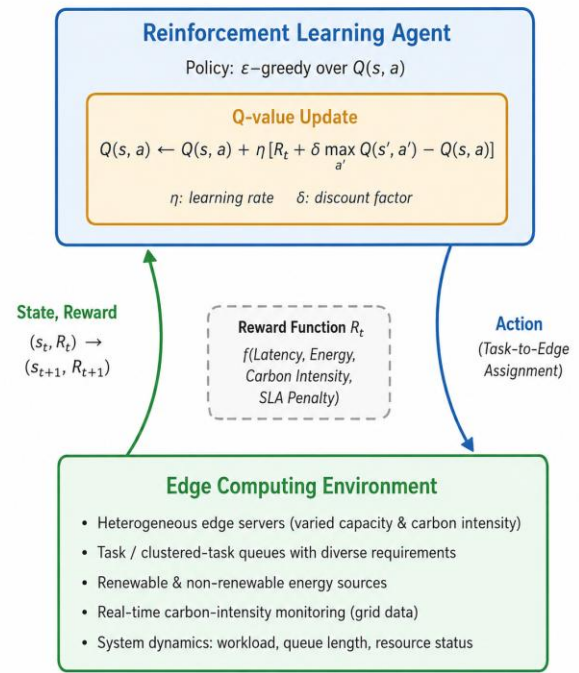


Figure 2: Closed-loop agent–environment interaction and task scheduling flow across edge and cloud layers.

C. Reliability and Security Controls

The system rejects commands with invalid schemas, unknown robots, expired timestamps, incompatible versions, or disallowed task types. Web and WebSocket connections require authenticated and encrypted transport before physical deployment. Secrets are supplied through deployment configuration rather than embedded in images. Health checks distinguish cloud unavailability from robot unavailability, and the edge continues local safety behavior when cloud services are lost.

VI. EVALUATION FRAMEWORK

The proposed system is an implementation blueprint; no fabricated latency, accuracy, or navigation results are reported. Evaluation should use fixed worlds, repeatable start and goal poses, controlled network conditions, and explicit software versions. Scenarios should include an empty environment, static obstacles, a narrow passage, replanning, delayed telemetry, cloud disconnection, backend restart, and dashboard reconnection.

A. Communication Metrics

Report median, 95th-percentile, and maximum end-to-end latency, together with jitter, message delivery rate, stale-message rate, reconnection time, and duplicate-task rate. Mean latency alone is insufficient because rare delays can dominate operator experience and cloud-edge coordination. Clock

synchronization assumptions or round-trip measurement methods must be documented.

B. Mapping and Navigation Metrics

Mapping evaluation should consider occupied/free-cell agreement, map completeness, structural consistency, and repeatability. Navigation evaluation should report goal success rate, time to goal, executed path length, collision count, recovery count, and localization error. If L_p is planned path length and L_e is executed path length for a successful run, path efficiency is:

$$\eta_p = L_p / L_e, \quad 0 < \eta_p \leq 1. \quad (2)$$

C. Resource and Failure Metrics

CPU, memory, network traffic, update frequency, and storage growth should be measured separately for edge and cloud components. Reliability tests should verify safe edge operation during cloud loss, rejection of stale commands, consistent task-state transitions, dashboard recovery, map availability, and database persistence after service restart. A successful failure test may result in a safe fallback rather than completed personalization or remote control.

VII. EXPECTED OUTCOMES, BENEFITS, AND LIMITATIONS

The expected outcome is a virtual robot that maps an environment, localizes, accepts navigation goals, plans paths, avoids obstacles, and reports status while cloud services manage tasks, records, monitoring, and visualization. The platform should demonstrate that low-latency control can remain at the edge while cloud components add observability and scalability. Simulation lowers entry cost for students and reduces the need for repeated hardware experiments during early development.

The system can support education and later research in warehouse automation, hospital service robots, industrial monitoring, smart agriculture, campus surveillance, and disaster response. Its modular interfaces also provide practical exposure to distributed systems, robotics, APIs, real-time communication, databases, containers, orchestration, testing, and DevOps practices.

The principal limitation is that a virtual robot does not reproduce all physical effects. Sensor noise, wheel slip, calibration drift, motor saturation, mechanical failure, battery behavior, and real radio interference require later hardware validation. The first phase targets one robot and local Minikube deployment; production AWS deployment and fleet coordination are future work. Integration complexity and limited development-machine resources may also affect simulation speed and repeatability.

VIII. FUTURE SCOPE

Future work can replace simulated sensors with physical LiDAR, camera, IMU, encoders, and embedded compute while preserving ROS 2 and cloud interfaces. Cloud components can migrate to AWS using managed Kubernetes, databases, observability, identity, and secure device provisioning. A multi-robot extension can add shared maps, task allocation, traffic management, conflict resolution, and fleet-level monitoring.

Additional research directions include semantic mapping, object detection, anomaly detection, predictive maintenance, battery-aware task planning, network-quality-aware service placement, and resilient offline operation. Large-scale evaluation should compare local-only, cloud-only, and cloud-edge configurations under identical tasks and explicitly quantify the performance, reliability, cost, and safety trade-offs.

IX. CONCLUSION

This review presents a cloud-edge collaborative architecture for mapping and control of a virtual mobile robot. ROS 2 Jazzy, Gazebo Harmonic, SLAM, localization, and Nav2 form a realistic, low-cost edge workflow. FastAPI, WebSockets, Redis, PostgreSQL, and React.js provide tasks, telemetry, storage, maps, and monitoring. Docker and Kubernetes/Minikube make the services modular and reproducible.

The central design principle is that cloud scalability must not replace local safety. Sensor handling, state estimation, obstacle response, navigation, and velocity control remain near the robot, while suitable monitoring and data functions are centralized. The architecture diagram, implementation methodology, and evaluation plan provide a foundation for academic validation and a controlled transition to physical robots, managed cloud infrastructure, AI perception, and multi-robot systems.

X REFERENCE

- [1] M. Quigley et al., "ROS: An Open-Source Robot Operating System," in Proc. IEEE ICRA Workshop on Open Source Software, Kobe, Japan, 2009.
- [2] Open Robotics, "ROS 2 Jazzy Documentation," 2026. [Online]. Available: <https://docs.ros.org/en/jazzy/>
- [3] Open Robotics, "Gazebo Harmonic Documentation," 2026. [Online]. Available: <https://gazebosim.org/docs/harmonic/>
- [4] Open Navigation LLC, "Nav2 Documentation," 2026. [Online]. Available: <https://docs.nav2.org/>

- [5] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," IEEE Robot. Autom. Mag., vol. 13, no. 2, pp. 99-110, 2006.
- [6] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," IEEE Trans. Robot., vol. 23, no. 1, pp. 34-46, 2007.
- [7] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in Proc. AAAI, 1999, pp. 343-349.
- [8] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," in Proc. IEEE/RSJ IROS, 2020, pp. 2718-2725.
- [9] M. Satyanarayanan, "The Emergence of Edge Computing," Computer, vol. 50, no. 1, pp. 30-39, 2017.
- [10] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet Things J., vol. 3, no. 5, pp. 637-646, 2016.
- [11] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A Survey of Research on Cloud Robotics and Automation," IEEE Trans. Autom. Sci. Eng., vol. 12, no. 2, pp. 398-409, 2015.
- [12] K. Goldberg and B. Kehoe, "Cloud Robotics and Automation: A Survey of Related Work," EECS Dept., Univ. California, Berkeley, Tech. Rep. UCB/EECS-2013-5, 2013.
- [13] S. Ramirez, "FastAPI Documentation," 2026. [Online]. Available: <https://fastapi.tiangolo.com/>
- [14] Meta Platforms, Inc., "React Documentation," 2026. [Online]. Available: <https://react.dev/>
- [15] I. Fette and A. Melnikov, "The WebSocket Protocol," IETF RFC 6455, Dec. 2011.
- [16] PostgreSQL Global Development Group, "PostgreSQL Documentation," 2026. [Online]. Available: <https://www.postgresql.org/docs/>
- [17] Redis Ltd., "Redis Documentation," 2026. [Online]. Available: <https://redis.io/docs/>
- [18] Docker, Inc., "Docker Documentation," 2026. [Online]. Available: <https://docs.docker.com/>
- [19] The Kubernetes Authors, "Kubernetes Documentation," 2026. [Online]. Available: <https://kubernetes.io/docs/>
- [20] Kubernetes SIGs, "Minikube Documentation," 2026. [Online]. Available: <https://minikube.sigs.k8s.io/docs/>