Nav2 for Autonomous Mobile Robots: An Implementation Oriented Survey of Architecture, Components, and Practical Limitations

Lim Yun Ler [1], Thangavel Bhuvaneswari [1,*], Min Thu Soe [1], and Muhammad Bin Hishamuddin [2]

[1] Centre for Advanced Robotics, Faculty of Engineering and Technology, Multimedia University, Melaka, Malaysia
[2] Centre of Robotics and Sensing Technologies, TM Research and Development Sdn Bhd, Cyberjaya, Malaysia
Email: yunlerlim01@gmail.com (L.Y.L.); t.bhuvaneswari@mmu.edu.my (T.B.); stmin@mmu.edu.my (M.T.S.); muhammad@tmrnd.com.my (M.B.H.)
*Corresponding author

***Abstract*—The Robot Operating System 2 (ROS 2) Navigation Stack (Nav2) has become the dominant framework for autonomous mobile robot navigation, yet practitioners encounter significant difficulty translating its architectural documentation into reliable real-world deployments. This paper presents a practitioner-oriented survey of Nav2's architecture, core components, and known implementation challenges, drawing on representative studies published between 2020 and 2025. Rather than claiming exhaustive literature coverage, this survey consolidates architectural knowledge by spanning Simultaneous Localization and Mapping (SLAM) and localization, path planning, costmap design and behavior trees, and documents recurring implementation experiences reported across diverse indoor robotic platforms. Key deployment challenges identified include odometry drift under sensor degradation, parameter tuning sensitivity, dynamic obstacle handling limitations, and real-time processing latency in distributed ROS 2 setups. The findings are intended to serve as a practical reference for engineers and researchers initiating Nav2-based indoor navigation projects, providing both technical grounding and awareness of where current implementations fall short.**

***Keywords*—autonomous navigation, costmap architecture, Robot Operating System 2 (ROS 2) Navigation Stack (Nav2), path planning, Simultaneous Localization and Mapping (SLAM) and localization**

I. Introduction

Autonomous navigation empowers mobile robots to perceive their surroundings, localize themselves within a map and plan safe paths to targets capabilities essential for applications from last-mile delivery and service robots to warehouse automation. As environments grow more dynamic and unstructured, navigation systems must be robust, flexible, and capable of real-time adaptation.

The original Robot Operating System 1 (ROS 1) Navigation Stack (Nav2) powered many pioneering projects, but its monolithic design, limited dynamic obstacle handling, and non-deterministic communication posed challenges for multi-robot coordination and real-time performance [1]. ROS 2 overcomes these limitations by adopting a Data Distribution Service (DDS) based middleware layer, managed lifecycles, and a plugin-driven architecture that supports behavior trees, secure execution, and easier integration of new algorithms. Building on these advances, the Nav2 stack offers modular planners, controllers, costmap layers, and recovery behaviors that can be customized or extended via standard ROS 2 interfaces [2, 3].

This paper aims to address this gap by presenting a consolidated survey of the Nav2 framework, including its architecture and core modules, with particular emphasis on mapping, localization, and path planning. Rather than a systematic enumeration of the literature, it draws on representative implementation studies to document practical experiences, recurring deployment challenges, and observed performance characteristics. The objective is to provide a technically grounded and practically useful reference for engineers and researchers, particularly those initiating ROS 2-based indoor navigation projects. To support this, the following section introduces the Nav2 architecture and its core design principles, which establish the foundation for all component-level and challenge-oriented discussion that follows.

II. Overview of ROS 2 Navigation Framework

ROS 2 Navigation (Nav2) is the official navigation framework in the ROS 2 ecosystem, designed to enable mobile robots to autonomously navigate in known or unknown environments. It provides a complete, modular, and extensible system for robotic navigation. Nav2 is the successor to the ROS 1 move_base navigation stack and addresses many of the architectural limitations of its predecessor as shown in Table I [4, 5].

Manuscript received December 30, 2025; revised March 9, 2026; accepted May 7, 2026; September 24, 2026.

doi: 10.18178/ijmerr.15.5.496-513

A. Architecture and Design Philosophy of Nav2

The evolution of robotic navigation has steadily transitioned from monolithic, ad hoc systems to frameworks characterized by modularity, extensibility, and robustness. Nav2 embodies this paradigm shift as the next-generation successor to the ROS Navigation Stack. It is crafted not merely as a collection of algorithms but as a comprehensive framework that integrates design philosophy with agile, service-oriented architecture. This design ensures that developers and researchers can rapidly deploy, customize, and enhance navigation behavior across diverse robotic platforms with stack in Table II.

TABLE I. COMPARISON OF ARCHITECTURAL FEATURES BETWEEN THE ROS 1 MOVE_BASE AND ROS 2 NAV2 NAVIGATION FRAMEWORKS

Feature	ROS 1 Move_Base	ROS 2 Nav2
Architecture	Monolithic	Modular, Plugin-based
Communication Middleware	ROS TCP	DDS-based Real-Time
Execution Flow	Finite State Machine	Behavior Trees
Lifecycle Management	Not supported	Lifecycle Nodes (Managed States)
Costmap Plugin Support	Basic	Highly extensible

TABLE II. PRINCIPAL COMPONENTS OF THE NAV2 STACK AND THEIR FUNCTIONAL ROLES

Key Components	Description
Planner Server	This module is responsible for computing global paths based on input from costmaps and environmental data. It efficiently maps out optimal trajectories from a start point to a given goal, integrating static and dynamic obstacle information.
Execution Flow	The Controller Server converts this path into velocity commands to provide fluid, actionable motion, while the Planner Server describes the trajectory. To provide precise path following even in the face of dynamic impediments or unpredictable environmental conditions, it continuously modifies these orders in response to sensor feedback.
Communication Middleware	Serving as the robot's environmental model, costmaps are continually updated grids that reflect obstacle positions and terrain complexity. They provide essential data for both global planning and local obstacle avoidance.
Lifecycle Nodes	Nav2 utilizes managed nodes that follow a lifecycle pattern. This approach establishes controlled startup, execution, and shutdown procedures, ensuring that each component transitions gracefully between states. Such systematic handling minimizes risks associated with abrupt failures or unexpected interactions between components.

At its core, Nav2's design philosophy is driven by three interlocking principles: modularity, flexibility, and robustness. The framework advocates decomposing the navigation task into discrete functional components. This modular approach minimizes code duplication and encourages the reuse of well-defined interfaces, allowing components such as path planning, control, and recovery behaviors to be independently developed and refined. By decoupling these functions, Nav2 empowers developers to tailor individual modules without risking regressions in other parts of the system. This not only streamlines testing and development but also fosters a vibrant ecosystem where third-party contributions can seamlessly enhance or replace specific functionalities.

In addition, Nav2 embraces a plug-and-play mentality. It is engineered to support various robot kinematics, from differential-drive and holonomic platforms to more complex Ackermann and legged configurations. Such flexibility is crucial for ensuring that the same core framework can be deployed in a multitude of environments, each with its unique challenges. In doing so, Nav2 emphasizes a forward-looking approach where the architecture remains resilient in the face of evolving hardware and operational requirements.

In ROS 1, the monolithic move_base architecture tightly couples global planning, local control, and recovery behaviors within a single execution pipeline. As a result, failure in one component can propagate and stall the entire navigation process. In contrast, Nav2 decomposes these functions into lifecycle-managed nodes with well-defined interfaces. This modular separation enables partial system recovery. For instance, controller or planner servers can be reset independently without restarting the full navigation stack. Empirical studies such as Refs. [6, 7] further demonstrate improved fault isolation and system resilience in distributed Nav2 deployments, particularly in multi-robot scenarios.

B. Architecture of Nav2

Built on the modular, lifecycle-managed foundation described above, Nav2's high-level control logic is orchestrated through behavior trees, which provide the hierarchical decision-making structure that coordinates planning, control, and recovery operations at runtime. Behavior trees offer a hierarchical, graphical representation of decision-making processes, where individual nodes encapsulate specific actions or conditions. This decomposition facilitates not only clear organization and debugging but also the dynamic reordering of tasks in response to real-time environmental changes. The result is a navigation system that can combine sequential and parallel logic, thereby enhancing both efficiency and reliability during mission-critical operations. The high-level architecture is shown in Fig. 1.

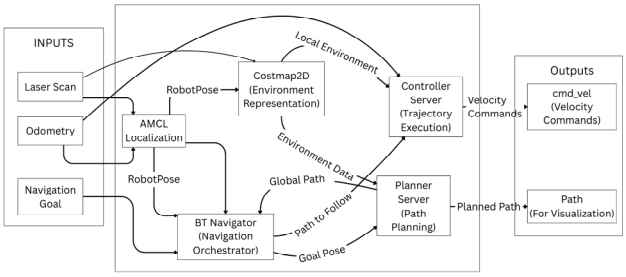

Fig. 1. High-level architectural overview of the Nav2 stack.

As shown in Table III, these modules are orchestrated by a high-level Behavior Tree Navigator, which

effectively manages the overall navigation logic. This hierarchical coordination allows for the integration of recovery strategies when unforeseen obstacles or errors are encountered, thereby maintaining operational continuity.

TABLE III. TYPICAL LIFECYCLE PROGRESSION IN NAV2

State/Transition	Node Status Type	Description
Unconfigured	Primary State	Initial state. Node exists but has not yet set up resources or parameters.
Configuring	Transition State	Prepares the node (e.g., loads parameters, sets up publishers/subscribers).
Inactive	Primary State	Node is initialized but not processing data or interacting with other nodes.
Activating	Transition State	Final preparation before the node becomes fully functional.
Active	Primary State	Node is fully operational—processing data, communicating, and executing tasks.
Deactivating	Transition State	Node stops processing data and disables its interfaces.
CleaningUp	Transition State	Node releases resources and resets state.
ShuttingDown	Transition State	Final resource cleanup before node is terminated.
Finalized	Primary State	Node is completely shut down and ready to be destroyed.
ErrorProcessing	Transition State	Node is handling failure. May attempt recovery or trigger shutdown.

An infrastructure component that underpins all spatial computation in Nav2 but is rarely discussed explicitly in deployment guides is the ROS 2 Transform Tree, managed by the tf2 library. The tf2 system maintains a directed acyclic graph of coordinate frame relationships across the robot, allowing any component to express spatial quantities include sensor observations, robot poses, planned paths, and velocity commands in any reference frame through dynamic lookup and composition of intermediate transforms.

Every spatially aware operation in Nav2 depends on tf2, costmap sensor ingestion transforms observations from sensor frames into costmap coordinates, Adaptive Monte Carlo Localization (AMCL) particle evaluation transforms expected scan patterns from map frame into laser frame, and the controller server expresses velocity commands relative to the robot's base frame. The tf2 tree is therefore not a peripheral utility but the spatial coordination infrastructure that enables Nav2's modular multi-component architecture to function coherently. For simple robot configurations with few sensors and no articulated joints, tf2 overhead is negligible. For complex robots with articulated structures, multiple sensor heads, or dynamically reconfiguring kinematic trees, tf2 management becomes a non-trivial deployment consideration that directly affects navigation reliability and latency.

C. Lifecycle Management

One of the most significant architectural improvements in ROS 2, adopted by the Nav2 framework, is the concept of Lifecycle Nodes, also known as Managed Nodes, as illustrated in Fig. 2. These nodes follow a structured state machine model with defined states and transitions, enabling better control over initialization, execution, error recovery, and shutdown. Starting with ROS 2 Humble, the APIs and toolchain support made lifecycle management robust and practical.

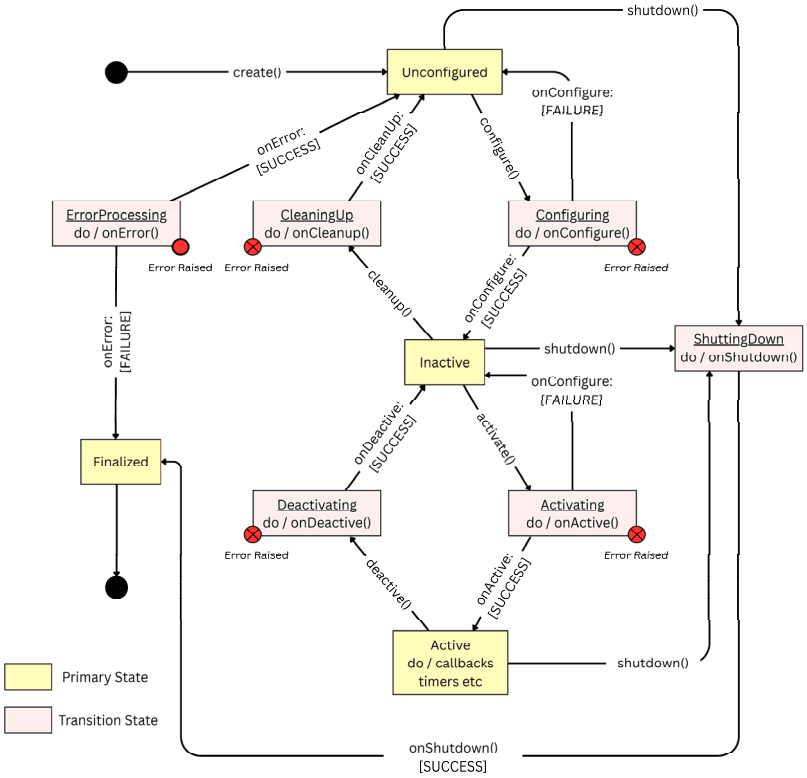

Fig. 2. Lifecycle action triggers each transition.

Beyond architectural benefits, Macenski *et al.* [8] demonstrate that Lifecycle Nodes also deliver clear performance advantages, CPU utilization is significantly lower compared to standard multi-process nodes. Cacciabue *et al.* [9] show that deactivated Lifecycle Nodes execute less code, which reduces CPU usage and battery consumption compared to redeployment. This shows that Lifecycle Nodes not only improve reliability and manageability but also enhance efficiency in resource-constrained robotic systems.

III. Core Components of ROS 2 Navigation

A structured literature search was conducted across major academic databases, including IEEE Xplore, Scopus, ACM Digital Library, and Google Scholar. Search queries combined terms such as "ROS 2 navigation", "Nav2 autonomous mobile robot", "ROS 2 Simultaneous Localization and Mapping (SLAM) localization", and "Nav2 path planning". The search was limited to peer-reviewed publications from 2020 to 2025. Studies were included if they implemented or evaluated Nav2 or its core components in simulation or real-world settings and reported navigation performance metrics. Works focusing exclusively on ROS 1, hardware design without navigation software integration, or non-English publications were excluded.

The studies cited throughout this section are not intended as an exhaustive census of literature. They were selected to illustrate the range of implementation approaches, sensor configurations, and performance outcomes encountered in Nav2-based navigation, and serve as representative examples from which practical lessons are drawn where shown in Tables IV and V.

TABLE IV. Summary of SLAM and Localization Configurations Reported in Surveyed Studies (2020–2025)

Paper	Sensor Suite	SLAM Methods	Localization Methods	Key Mapping Result	Key Localization Result
[10]	-	Gmapping	AMCL	Success rate of 92.6% in static environments	Average path length: 249.6 cm Average travel time: 34.6 s Static: 92.6% + 1.8 collision count Dynamic: 92% + intercollision count 1.2
[11]	Odometry Sensors, Inertial Measurement Unit (IMU) Fusion, LiDAR, RGB-D Camera	-	AMCL	-	Trigger alarms when approaching within defined thresholds (<50 cm)
[12]	IMU Fusion, LiDAR, RGB Camera	Grid-Based FastSLAM	AMCL	97% mapping coverage within 15 min; deviation from ground truth map within 5%.	Average localization error < 2%
[13]	Encoder Sensor, LiDAR	Cartographer (optimized)	AMCL - tuned for better particle filtering using KLD sampling.	Visually improved maps post-optimization with reduced drift and clearer obstacle boundaries.	Improved path-following and obstacle negotiation after controller_server tuning
[14]	RGB Camera	Grid-based Mapping	Custom method based on background subtraction and intruder detection using an overhead camera.	Mapping is done visually from overhead camera input.	Successful goal reaching and path following is demonstrated visually in maze-solving tests
[15]	Encoder Sensor, LiDAR	Slam-Toolbox	Slam-Toolbox	-	Dijkstra + RPP resulted in lower deviation and smoother path following, whereas A* + Dynamic Window Approach (DWA) exhibited higher deviations with occasional sharp turns.
[16]	LiDAR, wheel encoders	Cartographer	Cartographer's localization	-	At 0.6 m/s, the success rate was 82% for DWB and 96% for DWB + DOL, while at 0.8 m/s it decreased to 43.3% for DWB but remained higher at 86.7% for DWB + DOL.
[17]	Odometry, 9-aixs IMU, LiDAR, Camera Ultrasonic Sensor	Cartographer	Extended Kalman Filter (EKF) Fusion via Robot_Localization	Expected vs after EKF: 5 m vs 5.007 m, 20 m vs 20.307 m	measured errors <0.2 m
[18]	Optocoupler odometer, LiDAR, Wide-Angle Camera	SLAM Toolbox	SLAM Toolbox	Visual inspection showed highly detailed indoor maps with clear obstacle boundaries.	irrigation success rate = 29%
[19]	Wheel encoder, IMU, RGB-D Camera, Deep Learning-Based	SLAM Toolbox	Costmap based AMCL	-	keeps ≥3.0 m aways from person

	Perception, Embedded GPU				
[20]	Wheel encoders, IMU, LiDAR, RGB-D Camera, Ublox GNSS receiver	-	AMCL	-	Grass is correctly identified as traversable, resulting in shorter, more direct paths with plugin
[21]	Encoder, IMU, RGB-D Camera	SLAM Toolbox	AMCL	-	RB1: 3.3 Recoveries per mile TIAGo: 5.3 Recoveries per mile
[22]	RGB-D camera + scanner	vSLAM using FCN_ResNet50 for segmentation.	Camera-based semantic segmentation + ROS 2 Nav2 stack for local/global planning.	Enhanced by CNN segmentation + Canny edge fallback; improves obstacle edge detection.	Real-time updates reduce collision, fewer failures and shorter mission times compared to pure A*.
[23]	RGB-D, 2D/3D LiDAR, IMU, GPS	Integrated via ROS 2 Nav2; planners tested include learning-based and classical.	Nav2py plugin bridges Python planners with ROS 2; raw sensor data used directly.	Scenario generator produces semantically rich maps	Benchmarks show learning-based planners outperform classical in social metrics, but classical planners are more consistent globally.
[24]	3D LiDAR + RGB cameras + IMU + wheel odometry.	LiDAR-based SLAM + Voxblox TSDF mapping.	EKF fusion of IMU + odometry; loop closure with NDT.	Dense colored mesh reconstruction; 90% of points within 0.54 m of ground truth.	RPE ~0.28 m, ATE ~4–5 m depending on dataset; comparable to state-of-the-art SLAM.
[25]	LiDAR, IMU, Raspberry Pi camera.	LiDAR-based SLAM + ROS 2 navigation stack	LiDAR + odometry fusion; autonomous ground-to-air switching.	Focus on inspection tasks; maps generated via LiDAR scans.	Reliable transitions between aerial/ground modes; validated in indoor inspection scenarios.
[26]	GNSS with RTK, LiDAR, camera	ROS 2-based; Hybrid A*, DWA, Elastic Band used for trajectory planning.	GNSS + odometry + LiDAR fusion.	Map built from point clouds in RViz; cones placed with centimeter precision.	RTK GNSS ensures cm-level accuracy, Hybrid A* improves trajectory smoothness.
[27]	2D LiDAR + dual depth cameras.	SLAM Toolbox + RTAB-Map	EKF + AMCL particle filter.	2D maps efficient for navigation; 3D maps richer in environmental context.	EKF improves robustness; AMCL maintains pose accuracy even in cluttered corridors.
[28]	Velodyne VLP-16 LiDAR.	A* global planner + local planners DWB, MPPI, RPP	Motion capture + ROS 2 odometry.	Costmap-based mapping; static/dynamic obstacle scenarios.	MPPI best balance, DWB fastest but closer to obstacles, RPP smooth but weaker in dynamic settings.
[29]	Velodyne HDL-32E LiDAR + encoders.	SLAM Toolbox + Nav2 costmaps.	EKF + MoCap corrections	Layered costmaps	Reliable re-planning in dynamic environments, drift correction via loop closure.

TABLE V. SUMMARY OF GLOBAL AND LOCAL PATH PLANNING CONFIGURATIONS AND NAVIGATION PERFORMANCE REPORTED IN SURVEYED STUDIES (2020–2025)

Paper	Global Planner	Local Planner	Navigation Accuracy
[10]	-	DWA	Average path length: 249.6 cm Average travel time: 34.6 s Static: 92.6% + 1.8 collision count Dynamic: 92% + intercollision count 1.2
[11]	-	Includes alarm system for collision, stability, and failure detection.	Trigger alarms when approaching <50 cm, with increased alerts below 25 cm
[13]	Standard Nav2 planning architecture	Tuned through controller_server	Improved path-following and obstacle negotiation after controller_server tuning
[14]	A*, Dijkstra's, and DFS	Orientation detection and motion execution logic based on camera feedback were used for local adjustments.	Successful goal reaching and path following is demonstrated visually in maze-solving tests.
[15]	A* and Dijkstra	RPP and DWA	1. Dijkstra + RPP had lower deviation and smoother path following. 2. A* + DWA showed higher deviations and occasional sharp turns.
[16]	NavFn Planner (via nav2_navfn_planner/NavPlanner)	DWB Controller DWB + DOL combination for enhanced local path planning	In 0.6 m/s: Total successful navigation: 82% (DWB), 96% (DWB+DOL) In 0.8 m/s: Total successful navigation: 43.3% (DWB), 86.7% (DWB+DOL)
[17]	Nav2 Global Planning Module	Nav2 Local Planning Module	The vehicle reaches its target point with high precision. Map Comparison: Mapped dimensions suggest that the final positioning of the vehicle is accurate for the intended indoor delivery tasks.
[18]	Nav2's default global planner	Nav2's DWB Controller.	Irrigation success rate = 29%
[19]	nav2_navfn_planner/NavfnPlanner	dwb_core::DWBLocalPlanner	keeps ≥3.0 m aways from person
[20]	NavFnPlanner	DWBLocalPlanner	Without plugin: Robot perceives tall grass as obstacles, leading to detours.

			With plugin: Grass is correctly identified as traversable, resulting in shorter, more direct paths. Real-world orchard tests confirm practical improvements in path planning.
[21]	A* planner	TEB	RB1: 3.3 Recoveries per mile TIAGo: 5.3 Recoveries per mile
[22]	**A*** planner	CNN-based obstacle segmentation + reactive avoidance node	CNN segmentation reduced collisions and improved mission completion time. In Scene 2: A* alone took ~155 s, while A*+CNN took ~70 s. In Scene 3: A* alone failed with collisions (236 s), while A*+CNN recovered automatically (101 s).
[23]	foundation models, methods utilizing multi-modal inputs and control theory-based algorithms.	Social navigation controllers (learning-based + classical)	Learning-based planners excelled in social metrics. Classical planners more consistent in global metrics but aggressive near pedestrians. Benchmark showed ML planners outperform in crowded, dynamic social contexts.
[24]	LiDAR-SLAM + Voxblox TSDF mapping	-	Achieved 0.27–0.28 m RPE on Newer College dataset, comparable to SLAM. 90% of reconstructed points within 0.54 m error of ground truth. Strong mapping accuracy but not focused on navigation control.
[25]	global planner A*	Local planner for ground navigation + aerial mode switching	Hybrid robot achieved seamless aerial-ground transitions. LiDAR-based automatic switching improved autonomy. Energy profiling showed extended endurance by prioritizing ground mode.
[26]	Dijkstra, A*, Hybrid A*	DWA + Elastic Band	Hybrid A* gave smoother trajectories than A*. DWA handled dynamic obstacle avoidance. Elastic Band refined paths locally for collision-free navigation. Modular ROS 2 implementation allowed flexible switching.
[27]	SLAM Toolbox + RTAB-Map	DWB, DWB + DWB critics	2D SLAM efficient for navigation; 3D SLAM added richer context. Robot navigated narrow corridors robustly. Integration of 2D+3D improved adaptability in structured + unstructured environments.
[28]	A* (global planner for baseline path)	DWB, MPPI, RPP	MPPI: Best balance of safety + efficiency, smooth paths. DWB: Fast but closer to obstacles. RPP: Smooth but struggled in dynamic environments. Real-world tests confirmed MPPI most reliable in dynamic industrial settings.
[29]	Navfn (Dijkstra-based)	DWB Local Planner	Demonstrated Navigation2 deploy ability in industrial multi-robot setups. DWB critics tuned for obstacle avoidance and path alignment. Real-world + Gazebo tests showed robust re-planning when obstacles appeared.
[30]	Nav2 Global Planner Plugin	Nav2 Local Planner Plugin	The robot's path visualized in simulation and real-world tests generally aligns with the intended trajectory.
[31]	-	DWB, SCL, SFW	While the paper doesn't report standard metrics like mapping accuracy or docking accuracy, it uses 28+ custom metrics for benchmarking navigation quality.

A. SLAM and Localization

In autonomous mobile robotics, the ability to perceive and understand the environment is fundamental to safe and intelligent navigation. At the core of this capability lies Simultaneous Localization and Mapping (SLAM) and localization. These processes allow a robot to construct or use a map of an unknown environment and determine its position within that map in real time. In dynamic and unstructured environments, especially those shared with humans or containing moving obstacles, achieving accurate SLAM localization remains one of the most critical and challenging problems in robotics.

In the Nav2 framework, the SLAM and mapping components are typically the first subsystem activated in exploration-based navigation. It supplies a 2D occupancy grid map or 3D representation of the robot's operating environment, which is then used by the localization and planning components [32, 33].

For autonomous robots to function in situations when GPS is unavailable or pre-mapped data is out of date and it is crucial [34]. SLAM systems usually consist of a back-end optimization process that estimates the robot's trajectory and map consistency, frequently using methods like Extended Kalman Filter (EKF), particle filters, or pose graph optimization, and a front-end component that extracts features from sensor data such as LiDAR, cameras, or depth sensors. Cartographer and SLAM Toolbox are two popular SLAM solutions in ROS 2 [35, 36].

Localization allows the robot to identify its location on a map once the map has been constructed. The main method by which ROS 2 facilitates localization is AMCL [37], a particle filter-based algorithm that estimates the robot's position on a given map using laser scan data. Another popular choice is the Robot Localization package [38], which combines Extended or Unscented Kalman Filters to create more reliable and drift-resistant state estimations by combining input from several sensors, including GPS, encoders, and Inertial Measurement Units (IMUs). Several variables, including sensor accuracy, environmental structure, and parameter adjustment, affect localization accuracy. The accuracy of localization can be severely impacted by issues including odometry drift, sensor latency, and unclear environmental characteristics, particularly in dynamic settings that change often.

A specific limitation of AMCL arises in environments with highly repetitive or symmetric geometric features, such as long corridors or structured indoor layouts, where

similar sensor observations lead to particle ambiguity and incorrect convergence [39, 40]. In such cases, localization may drift or converge to incorrect poses despite stable odometry input. To mitigate this, increasing particle count and tuning resampling thresholds can improve hypothesis diversity, while integrating additional sensing modalities such as IMU or LiDAR-based odometry through EKF fusion provides complementary motion constraints. More advanced approaches incorporate semantic or visual features to reduce perceptual aliasing, although these introduce additional computational overhead.

The SLAM or localization output is fed into the larger navigation system in the Nav2 architecture, where it interacts with costmaps to integrate with both local and global planners. Using real-time sensor data, costmaps are layered 2D grid maps that show traversable space and obstacles. These layers consist of dynamic object updates, inflation zones, obstacle marking, and static maps. Smooth and safe path planning is ensured by the efficient integration of costmap layers and SLAM-generated maps, but this adds complexity, particularly when handling incomplete sensor data, dynamic obstructions, or computational constraints.

Several papers employ grid-based SLAM algorithms. For instance, Kashyap and Konathalapalli [10] reports that using GMapping results in a success rate of 92.6% in static environments, while combining FastSLAM with AMCL yields occupancy maps with 97% accuracy and localization errors under 2% [12]. These probabilistic methods generate detailed occupancy grids that underpin reliable navigation. Cartographer SLAM, though initially prone to drift and low-resolution maps, has shown improved obstacle definition and reduced drift once parameters such as submap overlap and scan density are tuned, enhancing global planning performance.

Beyond grids, several works extend LiDAR mapping with vision. Szwedka and Budzan [22] integrate RGB-D cameras and CNN segmentation to enrich obstacle detection, improving responsiveness but remaining sensitive to lighting conditions. In contrast, Serdel *et al.* [24] with the OLCMR architecture relies on LiDAR SLAM with NDT scan matching and EKF fusion, achieving localization errors of 0.28 m RPE and 4–5 m ATE, robust but less semantically rich. Kumar *et al.* [27] advance this discussion by integrating 2D LiDAR SLAM with 3D RGB-D SLAM, balancing efficient navigation with richer environmental context.

Localization methods converge on AMCL, valued for its particle-filter precision. Fine-tuning particle counts and thresholds can reduce errors and better practical localization accuracy [13], while Norbert *et al.* [26] demonstrate that GNSS with RTK refinement achieves centimeter-level accuracy outdoors. Ryalat *et al.* [25] highlight the challenge of maintaining localization across aerial and ground modes, where LiDAR ensures precision on the ground but IMU/odometry dominate in flight.

Taken together, these studies show that no single SLAM or localization technique suffices across all domains. Grid-based methods excel indoors; LiDAR SLAM ensures geometric fidelity in GNSS-denied settings; vision SLAM adds semantic awareness [41], and GNSS fusion dominates outdoors. The field is converging toward hybrid, multi-modal architectures that fuse LiDAR, vision, GNSS, and probabilistic filters to balance metric accuracy with contextual understanding, enabling robust navigation in diverse environments.

B. Path Planning

Another key feature of autonomous mobile robots is path planning, which enables a robot to move from a starting point to a target while avoiding obstacles and respecting its kinematic constraints. Path planning in Nav2 is organized into two levels: local planning and global planning. Based on a known or previously mapped environment, usually shown as a static 2D occupancy grid, the global planner creates an initial, ideal path. On the other hand, the local planner dynamically modifies this route in real-time by accounting for unexpected environmental changes like localization drift or moving obstacles.

Robots can function in both partially unknown and structured settings due to this layered architecture. While the local planner ensures immediate reaction and obstacle avoidance based on the robot's perception and motion feedback, the global path acts as a high-level route. This separation enhances flexibility and responsiveness, particularly in environments characterized by high uncertainty or frequent dynamic changes.

Although Nav2 offers a more robust and adaptable architecture than ROS 1, several challenges persist, including managing non-holonomic constraints, handling dynamic obstacles in real time, ensuring smooth trajectory generation, and establishing adaptive recovery behaviors upon navigation failure [42].

Nav2's path planning system leverages the pluginlib interface introduced in Section II, enabling runtime selection of planner algorithms through YAML configuration alone, without stack recompilation. This makes the system highly adaptable to different operational profiles, as detailed below.

Graph-based global planners such as Dijkstra, A*, Hybrid A*, and their Nav2 implementations are widely deployed because they guarantee collision-free routes and deterministic optimality in static maps [31, 35, 38]. NavFn applies Dijkstra or A* heuristics. Smac provides both a 2D A*-based planner, which assumes holonomic motion, and a Hybrid A* variant that accounts for robot kinematic constraints, enabling feasible trajectory generation for non-holonomic platforms such as differential-drive or Ackermann-steered robots. Theta* generates line-of-sight path segments. These planners achieve precise global routes, directly influencing metrics such as average path length and travel time. However, they remain brittle under odometry drift and actuator faults, and their reliance on static costmaps limits adaptability in dynamic multigoal scenarios. Studies comparing Dijkstra with A* variants show that while Dijkstra paired with smoother local controllers yields consistent trajectories, A* combined with Dynamic Window Approach (DWA) can produce erratic motion [15].

Optimization-based local planners are the most practically validated in ROS 2. The DWB controller, a modular extension of DWA, scores velocity commands based on path alignment, goal approach, and obstacle clearance. Its adaptability allows tuning trajectory critics to improve obstacle avoidance, yet it often cuts close to obstacles, reducing safety margins. The Regulated Pure Pursuit (RPP) controller emphasizes curvature-based velocity regulation, producing smoother paths but struggling with dynamic obstacles. The Timed Elastic Band (TEB) controller dynamically optimizes trajectories with respect to time, kinematics, and constraints, offering balanced accuracy at higher computational cost. Elbouhy *et al.* [28] show DWB excels in speed, MPPI balances safety and efficiency, and RPP yields smooth trajectories but triggers frequent recoveries. Navigation accuracy improves significantly when local costmaps incorporate dynamic obstacle layers: success rates rose from 43% to 87% when comparing DWB alone versus DWB+DOL [16].

Learning-based approaches are increasingly recognized as the most adaptable path planning techniques for dynamic environments. Unlike classical planners, they learn policies directly from data, enabling robots to anticipate and react to uncertainty. Szwedka and Budzan [22] and Shcherbyna *et al.* [23] showed CNNs and reinforcement learning outperform traditional controllers in social navigation and obstacle avoidance, though at higher computational cost. Moorthy *et al.* [43] advanced this by integrating Deep Deterministic Policy Gradient (DDPG) with ROS 2 and LiDAR, using reward functions to balance goal progress, collision avoidance, and trajectory smoothness directly improving navigation accuracy in dynamic obstacle scenarios. Complementing this, Cheng *et al.* [44] demonstrated that deep learning perception boosts object recognition accuracy above 90%, reducing misdetections that degrade path planning reliability. Together, these works highlight that learning-based navigation depends on the synergy between adaptive decision-making and accurate perception. The critical limitation remains computational overhead and training complexity, but the impact on navigation accuracy is clear: smoother trajectories, fewer collisions, and higher success rates in dynamic, multi-agent environments.

Finally, system-level constraints such as sensor latency directly affect navigation accuracy. ROS 2 improves LIDAR processing compared to ROS1, but camera latencies remain problematic [11], delaying local planner reactions to dynamic obstacles and reducing overall path-following precision.

C. Environmental Perception and Representation

In autonomous mobile robot navigation, the ability to perceive and interpret the surrounding environment is crucial. Within Nav2, this is achieved through a structured and extensible component known as the costmap architecture. The costmap serves as the robot's internal model of its operating environment, combining sensor inputs into a unified, spatially aware 2D grid. This model is used by planning, control, and monitoring modules to make intelligent, safe navigation decisions [45].

1) Costmap generation and management

The Nav2 costmap represents a dynamic, real-time model of the robot's environment, continuously updated based on sensor inputs. This allows the system to react to newly detected obstacles, terrain changes, or moving objects, as illustrated by the costmap architecture shown in Fig. 3 [46].

Each cell in the 2D grid is assigned a cost value reflecting the navigational risk:

- Free space (cost = 0): Safe areas;
- Lethal obstacles (maximum cost): Occupied or impassable regions;
- Inflated zones: Buffered margins around obstacles for safety clearance;
- Unknown space: Areas without sensor coverage.

To optimize computational efficiency, Nav2 employs multi-resolution costmaps. The local costmap, with higher resolution and faster update rates, focuses on the robot's immediate vicinity for responsive behavior. Meanwhile, the global costmap spans a broader area at lower resolution, supporting strategic long-range planning [47]:

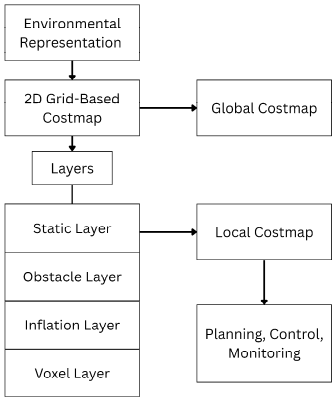

Fig. 3. Nav2 costmap architecture.

2) Costmap layer design

ROS 2 costmaps are built using a layered architecture, where each layer is a plugin managed by pluginlib. This design allows developers to independently include, configure, or extend specific functional layers. Standard layers include are described in Table VI [48].

Each layer operates independently and asynchronously, which allows for robust and efficient updates. The composite costmap is constantly rebuilt as new sensor data arrives, ensuring that the robot reacts to environmental changes with minimal latency.

To be noted, the Nav2 costmap consists of four main layers: static, obstacle, inflation, and voxel but the voxel layer is optional. This is because it is specifically designed for 3D obstacle representation, which is only needed when height-aware obstacle detection is required. If a robot operates in a flat environment with 2D LiDAR, the standard obstacle layer is sufficient, making the voxel layer unnecessary. However, for robots navigating in spaces with overhanging objects, varying terrain, or using 3D sensors like depth cameras, the voxel layer becomes

beneficial. Thus, its inclusion depends on the sensing setup and the complexity of the environment [49].

TABLE VI. COSTMAP LAYER DESIGN

Layer	Remarks
Static Layer	Loads a pre-existing occupancy map (e.g., from SLAM or a YAML map server) for static features like walls. Useful for large, well-known environments.
Obstacle Layer	Dynamically updates with data from live sensors like LiDAR, sonar, radar, or depth cameras. It detects obstacles in real-time and flags them in the costmap.
Inflation Layer	Adds a configurable buffer around detected obstacles, assigning progressively higher cost values outward from the obstacle boundary. This ensures the robot maintains safe distances.
Voxel Layer (Optional)	Enables 3D obstacle processing by maintaining a 3D voxel grid and projecting it into a 2D costmap. This is helpful when using depth sensors or stereo vision for volumetric environments, but it also introduces significantly higher memory consumption compared to standard 2D occupancy grids

D. Costmap Filters: Behavior-Modifying Spatial Constraints

A powerful feature introduced in Nav2 is the costmap filter system, which enables adaptive navigation behavior based on spatial semantics [50]. Unlike traditional costmap layers that primarily reflect obstacle presence, costmap filters overlay semantic annotations, allowing the robot to modify its behavior dynamically according to its location. These filters operate through filter masks, which function as spatial overlays on the costmap and introduce context-specific constraints without altering the underlying path planning algorithms. In Nav2, three primary costmap filter plugins are available, each serving a distinct function as listed in Table VII and visualized in Fig. 4.

TABLE VII. COSTMAP FILTER PLUGINS

Filter	Remarks
Keepout Filter	Enforces high-cost zones that robots must avoid. These are essential in environments where safety-critical regions—such as stairwells, hazardous machinery, or restricted zones—must be excluded from navigation.
Speed Filter	Constrains the robot's maximum velocity in specific areas. This ensures safer operations in shared spaces like corridors with pedestrian traffic or narrow indoor passages.
Binary Filter	Toggles robot behaviors or modes when entering or exiting predefined regions. For example, it can enable or disable certain actions based on location-based triggers.

These filters are implemented using annotated maps or dynamically generated masks, which are transformed into feature maps that modulate local cost values in real time. Importantly, the costmap filter architecture is modular and sensor-agnostic, remaining compatible with various perception systems such as 2D LiDAR or RGB-D cameras.

Additionally, each filter supports configurable parameters such as resolution, update rate, and cost modification rules, ensuring they are adaptable to both static map annotations and real-time sensor inputs. This design provides a robust mechanism for introducing high-level behavioral intelligence into mobile robots, making costmap filters essential in complex operational contexts from industrial floors and healthcare facilities to dynamic urban environments.

Fig. 4. Comparison of costmap layers and filters.

E. Sensor Fusion and Customization

The costmap layers in ROS 2 are designed to be sensor-agnostic, ensuring adaptability across different robotic platforms and configurations. Whether a robot relies on 2D LiDAR, 3D depth cameras, or radar arrays, sensor data can be processed and fused seamlessly into the costmap using pre-existing or custom layer plugins. This allows developers to tailor environmental perception to their specific operational needs.

To enhance accuracy and efficiency, raw sensor data undergoes pre-processing techniques such as filtering, clustering, and thresholding before being integrated into the costmap. This step helps eliminate noise, refine obstacle detection, and ensure smooth navigation in complex environments.

Beyond standard layers, developers can extend costmap capabilities by designing custom plugins for specialized tasks, including:

- Tracking moving obstacles, enabling dynamic adaptation to real-time environmental changes.
- Classifying object types, such as distinguishing between humans and static objects for safer interaction.
- Modifying cost values based on heuristics or behavioural models, optimizing navigation for specific use cases.

This modular and extensible approach is crucial for deploying autonomous robots in diverse real-world environments from hospitals and warehouses to outdoor terrains where environmental constraints and navigation demand vary significantly.

F. Impact of Costmap Architecture on Navigation Behavior

In Nav2, the costmap plays a central role in shaping how the robot perceives and reacts to its environment. Rather than being a static map, the costmap acts as a dynamic, real-time environmental model, constantly updated with data from multiple sensors. This evolving representation directly informs decision-making across the entire navigation stack, from global path planning to local trajectory control and safety monitoring.

Some paper also details how such architectures impact on their project [51, 52]. The costmap's information is accessed by several critical components within the ROS 2 Navigation Stack, each of which relies on its spatial representation to ensure safe and efficient robot behavior also had shown in Table VIII.

Proper configuration of costmap parameters is essential to balance reactivity, safety, and efficiency. Key parameters included are shown in Table IX.

TABLE VIII. COSTMAP INFLUENCE ACROSS NAVIGATION MODULES

Navigation Modules	Remarks
Global Planners	These planners use the global costmap to compute an overall path from the robot's current position to a target goal. The costmap enables the planner to identify collision-free routes, accounting for static and semi-static obstacles in the environment.
Local Planners	The local costmap is sampled at high frequency to adapt to the robot's immediate surroundings. As new obstacles—such as moving people or objects—are detected, the local planner updates the robot's trajectory to maintain smooth, reactive navigation without abrupt stops or collisions.
Collision Monitors and Recovery Behaviors	ROS 2 includes safety mechanisms like collision monitors, which continuously evaluate the robot's position relative to high-cost areas (e.g., occupied or inflated zones). If the robot gets too close to danger, recovery actions such as stopping, reversing, or replanning are triggered, enhancing resilience in uncertain environments.

TABLE IX. KEY COSTMAP CONFIGURATION PARAMETERS AND THEIR EFFECT ON NAVIGATION BEHAVIOR

Parameter	Function	Effect on Behavior
Update Frequency	How often the costmap is refreshed with new sensor data	High frequency increases reactivity; too high may cause noise or processing delays
Resolution	Size of each costmap cell (e.g., 0.05 m)	Finer resolution offers precision; coarser resolution reduces computational overhead
Inflation Radius	Distance around obstacles treated as unsafe for navigation	Larger radius adds safety margin but may block tight spaces
Obstacle Persistence	Duration a dynamic obstacle remains in the map after detection (in seconds)	Shorter time improves responsiveness; longer time improves stability against noise

G. Behavior Tree-Based Navigation and Recovery Behaviors

In the Nav2 framework, Behavior Trees (BTs) [53–55] serve as the primary mechanism for orchestrating navigation tasks. This architectural shift from the monolithic Finite State Machines (FSMs) [56] used in ROS 1 introduces a modular, extensible, and robust structure for coordinating planning, control, and recovery behaviors. BTs enable the navigation stack to make decisions dynamically, adapt to changing environments, and gracefully handle failures during navigation.

A BTs defines a hierarchical control structure that sequences planning, control, and recovery operations in a logical and flexible manner as shown in Fig. 5. Each node in the tree represents a behavior or decision, which can succeed, fail, or continue running where listed in Table X. These nodes are evaluated in order, enabling the robot to react intelligently to both success and failure conditions.

Recovery behavior selection becomes particularly consequential in human-populated environments. Standard Nav2 recovery nodes such as SpinRecovery, while effective in empty corridors, increase the robot's instantaneous swept footprint and may violate proxemic boundaries in dense spaces. ClearCostmap recovery carries the additional risk of erasing valid human obstacle markings, potentially generating paths that route through temporarily occupied regions. The Wait recovery node is

arguably the most socially appropriate default in high-traffic scenarios, allowing transient human flows to clear before re-planning is attempted. However, the surveyed literature does not provide direct comparative evaluation of recovery node performance stratified by human density or traffic pattern. Macenski *et al.* [21] documents experiments in dense, high-traffic campus corridors where Clear Costmap, Wait, and Spin were the most effective Behavior Tree nodes for stuck recovery, yet the study reports only aggregate recovery frequencies rather than stratified comparisons, reinforcing the gap that no work has systematically evaluated recovery node selection as an independent variable in human-shared environments.

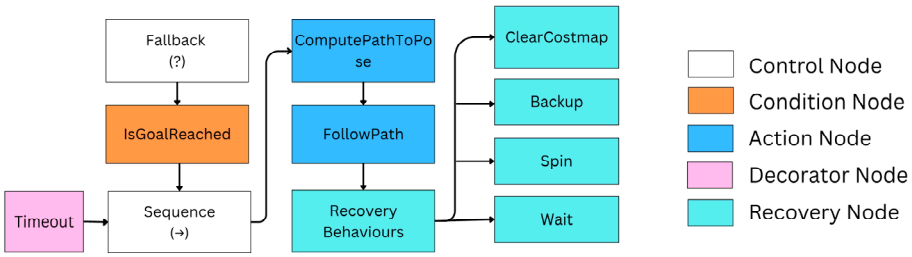

Fig. 5. Behavior trees flow.

TABLE X. COMMON BEHAVIOR TREE NODES IN NAV2

Node Type	Example Plugin	Purpose	Typical Usage
Control Node	Sequence, Fallback	Direct the flow of child nodes	Sequences of tasks or chooses alternatives
Action Node	NavigateToPose, FollowPath	Executes high-level navigation or control actions	Move base, follow global/local plan
Condition Node	IsGoalReached, IsStuck	Evaluates the condition to influence BT decisions	Check if the goal is reached or robot is stuck
Decorator Node	RetryUntilSuccessful	Modifies behavior of a child node	Retry failed tasks a set number of times
Recovery Node	ClearCostmap, SpinRecovery	Executes recovery behavior upon failure	Clear local/global maps, rotate in place

Although behavior trees provide flexible and modular recovery handling, their execution is not inherently guaranteed to terminate. Improper tree design, particularly with repeated fallback or retry decorators, can lead to infinite recovery loops. Formal validation of behavior tree correctness can be approached using methods such as finite-state abstraction, model checking, or temporal logic verification. However, such techniques are not commonly applied in current Nav2 implementations, where correctness is typically ensured through empirical testing and careful design of retry limits and termination conditions. Nevertheless, Zafar *et al.* [57] demonstrates that BT logic can be mathematically validated by translating behavior trees into Datalog rules and applying formal queries. This ensures termination of the analysis and allows detection of infinite recovery loops, providing a rigorous method to guarantee correctness beyond empirical testing.

IV. CHALLENGES

The components surveyed above include SLAM, localization, path planning, costmap layers, and behavior trees are collectively enable autonomous navigation within the Nav2 framework. However, their practical deployment reveals consistent limitations that cut across component boundaries. The following section examines these challenges systematically, drawing on the same body of implementation studies to identify where current Nav2-based systems fall short.

A. Sensor Reliability and Odometry Drift

Odometry drift remains the most consistently reported limitation in Nav2-based navigation systems. It originates from the cumulative integration of small errors in wheel encoder measurements, which become significant during long-duration operation, frequent turning, or traversal over uneven terrain. Without external correction, these errors lead to divergence between estimated and actual robot pose, resulting in costmap misalignment, degraded path tracking, and eventual navigation failure.

In Ref. [17], cumulative error reaches approximately 0.3 m over a 20 m indoor trajectory when relying on raw odometry, exceeding the tolerance required for precise goal alignment in constrained spaces. Even when overall success rates appear stable, as in Ref. [10], increased collision frequency in dynamic scenarios indicates underlying localization instability. More severe degradation is observed in unstructured environments: Fang *et al.* [18] reports only 29% task success when relying on low-resolution encoder feedback, while Nuhel *et al.* [14] shows that vision-only localization can fail entirely under poor lighting conditions.

The dominant mitigation strategy remains multi-sensor fusion via the EKF-based robot_localization framework introduced in Section III.A, which has been shown to reduce positional error to below 0.2 m over a 20 m indoor trajectory [17]. More advanced systems combine LiDAR SLAM, loop closure, and multi-modal fusion to further stabilize long-term localization [24]. These methods improve robustness across both structured and

unstructured environments by compensating for individual sensor weaknesses.

However, a key limitation persists: all surveyed approaches rely on static fusion configurations, where sensor weights and noise models are fixed during deployment. This assumption does not hold in real-world conditions, where sensor reliability can change dynamically due to wheel slip, surface variation, or environmental interference. As a result, degradation in one modality propagates through the fusion process without explicit detection or adaptation. The absence of adaptive or fault-aware fusion mechanisms highlights a fundamental gap in current Nav2-based systems, particularly for long-term autonomous operation in variable environments.

B. Real-Time Constraints in Multi-sensor and Distributed Systems

Robust Nav2 operation requires integrating heterogeneous sensor streams with differing update rates and data characteristics. LiDAR, IMU, encoders, and cameras operate at different frequencies and produce asynchronous data, making consistent state estimation and costmap generation inherently challenging. This mismatch introduces synchronization errors, where the fused state does not accurately reflect any single sensor observation at a given time.

Empirical studies show that fusion improves accuracy but introduces system complexity. Shcherbyna *et al.* [23] highlights that integrating multi-modal data improves evaluation performance but complicates system design and reproducibility. Similarly, Kumar *et al.* [27] demonstrates that combining 2D and 3D SLAM improves robustness in cluttered environments but requires careful coordination between sensing pipelines.

Current approaches rely on filtering frameworks such as EKF and modular costmap layers to combine sensor inputs. However, these methods assume consistent timing and reliability across sensors. In practice, variations in update rates and intermittent sensor degradation introduce inconsistencies that are not explicitly modelled.

As a result, sensor fusion in Nav2 remains accuracy-oriented rather than consistency aware, with limited mechanisms to manage asynchronous or degraded inputs during operation.

C. Dynamic Environment Limitations of Standard Planners

Nav2's default planners are fundamentally reactive, operating on instantaneous costmap representations without predicting future obstacle motion. This limits performance in environments with moving agents, where safe navigation requires anticipating changes rather than reacting to them.

As demonstrated in Section III.B, standard planners operating without dynamic obstacle layers show markedly lower success rates [16]. Comparative studies in Ref. [30] further show that different planners exhibit trade-offs between safety, efficiency, and smoothness, with no single approach performing consistently across scenarios.

Mitigation strategies include dynamic obstacle layers, social costmaps, and learning-based planners. While these improve performance, they are implemented as extensions rather than integrated components of the planning framework.

The Nav2 global planner is invoked reactively; it re-plans when the local controller reports a failure condition or when the current global path is invalidated. It does not continuously monitor costmap changes to pre-emptively re-plan. When dynamic obstacles inflate into the global costmap and block the planned path, the system depends on the local planner to navigate around them first. Only upon local planner failure does a global re-plan occur. This architectural decision trades responsiveness for computational efficiency. In environments with frequent or large-scale dynamic changes, this creates a structural latency where the global path remains stale until a failure is explicitly detected. Studies such as Refs. [25, 37] implicitly demonstrate this limitation through their reported recovery rates, though none directly instrument the re-planning trigger frequency. This represents a known gap that adaptive or predictive global planners such as Hybrid A* with rolling costmap windows partially address, as seen in Ref. [26].

D. Real-Time Communication Latency

Despite the real-time communication features of ROS 2, system-level latency remains a critical constraint in Nav2 deployments. Delays arise from sensor processing pipelines, inter-node communication, and multi-component architectures, resulting in outdated information being used for planning and control.

Studies such as Ref. [11] show that delayed perception updates reduce the effective reaction window for obstacle avoidance, while Horelican [29] demonstrates that distributed system architectures introduce additional communication delays that limit responsiveness. In multi-robot systems, shared communication channels further amplify latency effects.

Current mitigation strategies include QoS tuning, hardware acceleration, and architectural separation of processing tasks. However, these approaches are largely system-specific and do not provide guarantees on end-to-end responsiveness. A key limitation is the absence of formal latency modelling, where acceptable delays are defined relative to robot dynamics and safety requirements. Without this, system performance remains empirically tuned rather than analytically grounded.

Furthermore, Park *et al.* [58] analytically decomposes end-to-end delay into queuing, serialization, deserialization, and transport overheads, and demonstrates through experiments that DDS overhead becomes significant under high-frequency sensor streams like LiDAR.

E. Parameter Tuning and System-Level Integration Complexity

The flexibility of Nav2's modular architecture introduces a high-dimensional parameter space that significantly affects system performance. Navigation behavior depends on tightly coupled parameters across

costmaps, planners, and localization modules, making independent tuning ineffective.

Experimental results show that parameter configuration can dominate system performance. Liang *et al.* [13] demonstrates that SLAM performance can be substantially improved through parameter adjustment alone. Cross-platform studies in Ref. [21] further indicate that configurations do not generalize across different robots, leading to inconsistent outcomes.

Current practices rely on manual tuning, parameter sweeps, and benchmarking frameworks. Although effective in controlled settings, these methods are time-consuming and lack scalability. The absence of automated or transferable tuning methodologies indicates that Nav2 deployment remains configuration-dependent, limiting reproducibility and practical adoption across diverse platforms.

To address this limitation, recent research directions focus on reducing manual tuning through automated and adaptive approaches. Optimization-based methods, such as Bayesian optimization and evolutionary strategies, have been explored to systematically search for high-dimensional parameter spaces [59]. In parallel, learning-based approaches, including reinforcement learning, have been proposed to adapt navigation parameters dynamically based on environmental feedback [60]. These methods aim to reduce reliance on manual configuration and improve generalization across platforms. However, their integration into Nav2 remains limited, and challenges such as training cost, stability, and real-time constraints must be addressed before widespread adoption.

F. Social Navigation and Proxemic Constraints.

Nav2's default architecture treats moving humans as dynamic obstacles, cost-valued cells in the local costmap without modelling their spatial comfort expectations or behavioral intent. This representation is functionally adequate for collision avoidance but insufficient for social navigation, where the robot's trajectory must account for proxemic zones, approach angles, and velocity profiles near persons.

TABLE XI. SUMMARY OF METHODOLOGIES, IMPROVEMENT STRATEGIES, AND CHALLENGES ADDRESSED IN SURVEYED STUDIES (2020–2025)

Paper	Improvement Strategy Targeted	Challenges Addressed
[10]	SLAM-based mapping and multi-robot navigation evaluation using path metrics and collision analysis.	Sensor drift; multi-agent coordination; instability in complex environments
[11]	Multi-sensor SLAM with real-time monitoring and teleoperation support	Sensor noise; system latency; reliance on manual intervention.
[12]	FastSLAM with AMCL and real-time LiDAR-based obstacle avoidance.	Odometry drift; SLAM tuning sensitivity; dynamic obstacle instability
[31]	Social-aware navigation using behavior trees and human-interaction modeling	Lack of social metrics; human modeling complexity; interaction uncertainty.
[13]	Parameter optimization of SLAM, AMCL, and controller for improved localization and path stability	Map quality degradation; localization drift; goal instability
[14]	Vision-based localization with grid mapping and classical path planning	Visual noise sensitivity; high computation load; lack of redundancy.
[15]	Comparative evaluation of global and local planner combinations in dynamic environments	Planner trade-offs; coordination inconsistency; dynamic instability
[16]	Dynamic obstacle layer with Kalman prediction and adaptive costmap updates	Lack of native dynamic handling; tuning complexity; limited high-speed response
[17]	EKF-based sensor fusion with optimized SLAM and Nav2 configuration	Sensor inaccuracies; indoor mapping variability
[18]	SLAM and navigation tuning with vision-based object detection and wireless modular system	SLAM drift; encoder limitations; perception errors
[19]	Social costmap integration using RGB-D perception and deep learning	High computation demand; limited generalization; hardware constraints
[20]	Semantic segmentation-based costmap refinement for travers ability awareness	Sensor misclassification; synchronization issues; computation overhead
[21]	Modular Nav2 architecture with behavior trees and adaptive multi-algorithm planning	Non-holonomic constraints; localization ambiguity; human intervention need
[22]	Vision-enhanced navigation with CNN-based obstacle detection and reactive safety layer integrated with A*	Planner instability; collision risk in dynamic scenes; sensitivity to visual conditions and thresholds
[23]	Python-based Nav2 plugin framework with unified benchmarking and multi-simulator evaluation	Sim-to-real gap; lack of standardized benchmarks; integration barriers
[24]	LiDAR-IMU-odometry fusion with dense 3D mapping and real-time reconstruction	Calibration sensitivity; computational load; drift in large-scale environments
[25]	ROS 2-based hybrid ground–aerial navigation system with autonomous mode switching	Mode transition instability; energy trade-offs; integration complexity
[26]	Multi-layer planning pipeline with enhanced localization and perception	Localization sensitivity; planner trade-offs; dynamic obstacle handling
[27]	Hybrid 2D-3D SLAM integration with multi-sensor fusion in Nav2 framework	Drift in large environments; limited environmental representation; sensor uncertainty
[28]	Comparative optimization of Nav2 local planners with real-world validation	Dynamic obstacle handling; planner-specific limitations; parameter sensitivity
[29]	Multi-robot Nav2 framework with distributed coordination and DDS-based communication	Multi-robot coordination; communication reliability; scalability challenges
[30]	Nav2 integration with sensor fusion, real-time re-planning, and sim-to-real calibration	Dynamic obstacle handling; fusion inconsistency; sim-to-real gap

Proxemics theory defines concentric comfort zones around individuals: intimate (<0.4 m), personal (0.4–1.2 m), social (1.2–3.6 m), and public (>3.6 m) [61, 62]. Standard Nav2 inflation layers do not encode these zones differentially; the inflation radius applies uniformly to all obstacle types including humans. Addressing this requires custom social costmap layers that assign asymmetric, direction-sensitive cost fields around detected persons, as demonstrated in Ref. [19], which enforces a 3.0 m standoff distance using RGB-D perception integrated with a modified costmap layer.

Ref. [31] advances this further by introducing HuNavSim, a ROS 2 simulator with 28+ social navigation metrics covering human comfort, trajectory naturalness, and interaction safety. Its findings indicate that standard Nav2 benchmarking protocols are insufficient for evaluating social behavior, as they measure only goal success and collision count rather than comfort-relevant trajectory characteristics.

Ref. [23] provides the most comprehensive comparison, showing that learning-based planners, including CrowdNav++ and SacSon outperform classical Nav2 planners on social metrics in multi-agent environments. Classical planners achieve more consistent global navigation but exhibit aggressive velocity profiles near pedestrians, a behavior that is technically safe by collision metrics but socially inappropriate.

The fundamental limitation is architectural: social compliance in Nav2 is implemented as a post-hoc extension through custom layers and non-standard planners rather than as a first-class navigation objective. This means social behavior cannot be formally guaranteed and degrades when custom components are absent or misconfigured. As indoor robots increasingly operate in human-shared spaces, this represents one of the most practically significant gaps in current Nav2 deployments.

G. Cross-Cutting System Limitation

Across all identified challenges, a consistent pattern emerges existing Nav2-based solutions predominantly address system-level symptoms through sensor fusion, planner augmentation, and architectural optimization rather than underlying sources of error as shown in Table XI. Low-level uncertainties such as encoder resolution, actuator faults, and wheel terrain interaction are not explicitly modelled within the navigation pipeline. Consequently, improvements in perception and planning remain highly dependent on the quality of odometry inputs. This indicates a structural gap in current research, where robustness is pursued through increased system complexity rather than foundational reliability, highlighting the need for systematic investigation at the sensing and actuation level.

V. Implementation Recommendations for Indoor Nav2 Deployment

The following recommendations are grounded directly in the implementation experiences documented across the surveyed studies. They are organized by decision points in the order a practitioner typically encounters them when configuring a Nav2-based system: SLAM algorithm selection, localization strategy, path planner selection, costmap configuration, and recovery behavior design. These are not prescriptive rules but evidence-based starting points; practitioners should treat them as informed defaults and adjust based on their specific hardware, environment geometry and operational requirements.

A. SLAM Algorithm Selection for Indoor Environments

For known, structured indoor environments, SLAM Toolbox is the recommended default due to its native ROS 2 support, loop closure capability, and compatibility with long-term map reuse. Cartographer is preferable when submap-level precision is required, but its default parameters produce drift and low-resolution boundaries that require tuning of submap overlap and minimum scan density before deployment. Gmapping, while functional, is a ROS 1 port and does not benefit from ROS 2 lifecycle management; it is not recommended for new deployments.

B. Localization Strategy by Environment Type

For static or semi-static indoor environments: AMCL with tuned particle counts typically 500–2000 depending on map complexity is sufficient and computationally economical. For environments with moving people or frequent reconfiguration, EKF fusion via robot_localization combining LiDAR odometry and IMU is recommended. For outdoor or large-scale environments: GNSS with RTK correction achieves centimeter-level accuracy but requires open-sky conditions and is not applicable to indoor use.

A related challenge is the kidnapped robot problem, where the robot is abruptly displaced to an unknown location, causing the particle distribution to become inconsistent with the actual pose. Standard AMCL mitigates this through random particle injections and global localization recovery, although convergence can be slow depending on particle diversity and sensor distinctiveness.

C. Local Planner Selection by Operational Profile

Based on the comparative evidence available, DWB is appropriate when speed is the primary requirement and the environment is predominantly static. It consistently performs fastest but reduces obstacle clearance margins. MPPI is recommended for dynamic industrial or shared environments where both safety and efficiency matter. It demonstrates the best overall balance in real-world tests. RPP produces the smoothest trajectories in structured corridors but triggers frequent recoveries in dynamic scenarios and is not recommended as a standalone planner where moving obstacles are present. TEB handles non-holonomic constraints well and is suitable for tight space navigation but imposes the highest computational cost among standard Nav2 controllers.

For practitioners considering DRL as an alternative to DWB, the architectural distinction between augmentation and replacement has direct implications for deployment strategy. DRL augmentation specifically critic weight adaptation or residual correction architectures is accessible for near-term deployment and can be evaluated

incrementally against a DWB baseline without requiring complete revalidation of the navigation stack. Full DRL replacement requires comprehensive validation across the robot's complete operating envelope and explicit fallback mechanism design before production use.

The Nav2py bridge provides the most practical integration pathway for Python-based DRL policies but introduces inference latency overhead that must be characterized on target hardware before committing to a specific controller frequency. Practitioners should benchmark policy inference time on the deployment platform under realistic load conditions, including concurrent SLAM, costmap updates, and sensor processing before configuring the controller frequency, as computational contention under load may produce inference times substantially longer than isolated benchmarks suggest.

DWB remains the recommended default for static and semi-static indoor environments due to its predictable behavior, extensive deployment validation across the surveyed literature, and interpretable failure modes. DRL-based controllers should be considered for environments with complex dynamic obstacle patterns or social navigation requirements where DWB's performance ceiling has been empirically reached and the engineering resources for training pipeline development and policy validation are available.

D. Costmap Configuration Pitfalls

Three parameters are most frequently misconfigured in the implementations surveyed. First, inflation radius set too large blocks navigation in narrow corridors. Recent study report 0.6 m inflation radius consistently prevented traversal of a 0.7 m corridor, while reducing it to 0.3 m enabled successful passage and cut failure rates by 40%, highlighting the trade-off between safety margins and corridor accessibility [63]. Second, obstacle persistence set too high causes the robot to avoid locations where transient obstacles have already cleared, producing unnecessary replanning cycles. Third, update frequency set beyond the sensor's actual publication rate creates an illusion of reactivity while consuming processing resources without benefit.

Odometry drift also affects the consistency of the rolling costmap window, particularly during high-speed maneuvers. Since the local costmap is typically maintained in the odometry frame, accumulated pose error can cause misalignment between perceived obstacles and their mapped positions, leading to inaccurate obstacle inflation or transient "ghost" obstacles. This effect is amplified at higher velocities, where rapid motion increases the rate of drift accumulation and reduces the effectiveness of local obstacle avoidance.

VI. Conclusion

This survey has consolidated architectural and implementation knowledge for the Nav2 framework, drawing on representative studies across simulation and real-world indoor deployments from 2020 to 2025. Rather than providing an exhaustive assessment of the research landscape, it is intended as a practical starting point for engineers and researchers deploying Nav2 in indoor autonomous mobile robot applications. By analyzing recent studies that implemented Nav2 in both simulation and real-world environments, we identified the common techniques used for SLAM, localization, path planning, and obstacle avoidance. While Nav2 brings important improvements over ROS 1 such as better communication performance, modularity, and support for behavior trees, it still presents several challenges in practice. Many researchers report issues such as localization drift caused by sensor noise, difficulty in parameter tuning, and reduced performance in dynamic or cluttered environments. The transition from simulation to real-world application also continues to reveal gaps in system robustness and adaptability.

Another key observation is that many studies do not fully examine the behavioral logic behind Nav2's operation, including how behavior trees are structured, how fallback or recovery behaviors are triggered, and how these influence real-world reliability. This gap limits our understanding of how to design more resilient navigation systems.

Beyond the single-robot navigation challenges documented in this survey, a transition pathway from individual Nav2 deployments to coordinated fleet management systems warrants explicit roadmap guidance for practitioners planning multi-robot scaling. In such architectures, Nav2 functions as a local autonomy layer, while centralized or distributed fleet managers coordinate global objectives across multiple robots. Establishing standardized interfaces and improving interoperability between navigation and fleet-level decision systems remains an important direction for scalable real-world deployment.

Building on the limitations of static costmaps and predefined recovery behaviors, another emerging direction involves extending Nav2 beyond purely geometric representations toward learning-enhanced navigation frameworks. Semantic costmaps enriched by perception systems enable differentiation between static structures and dynamic agents, supporting more context-aware and socially compliant behaviors. In parallel, learning-based methods such as hybrid reinforcement learning offer the potential to replace static recovery nodes in behavior trees with adaptive strategies capable of handling complex and unpredictable scenarios. Together, these approaches aim to improve robustness and autonomy; however, challenges related to computational cost, real-time inference, training stability, and integration with existing Nav2 architectures remain significant.

Overall, this paper brings together the lessons and limitations found across current Nav2 research. It is intended to help researchers and developers understand what has already been done, what techniques have worked well, and where challenges remain, especially for those planning to use Nav2 in robot applications. Moving forward, we recommend further research on adaptive sensor fusion, improved tuning and diagnostics tools, and better benchmarking methods that reflect realistic indoor

scenarios. As Nav2 continues to evolve, it has strong potential to support the development of more dependable and intelligent mobile robots.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

LYL conducted the literature search, performed the critical analysis and synthesis of the reviewed studies, and drafted the entire manuscript. TB, MTS and MBH provided supervision, conceptual guidance, and critical review of the manuscript. All authors have read and approved the final manuscript.

FUNDING

This research was funded by TM Research and Development Sdn. Bhd. under project code RDTC/241127.

ACKNOWLEDGMENT

This work was supported by TM Research and Development Sdn. Bhd. The authors also acknowledge the academic guidance and manuscript review provided during the preparation of this paper.

REFERENCES

[1] M. Quigley *et al.*, "ROS: An open-source Robot Operating System," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, vol. 3, no. 2, 5.

[2] D. Portugal, R. P. Rocha, and J. P. Castilho, "Inquiring the robot operating system community on the state of adoption of the ROS 2 robotics middleware," *Int. J. Intell. Robot. Appl.*, vol. 9, no. 2, pp. 454–479, 2025. doi: 10.1007/s41315-024-00393-4

[3] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, Architecture, and Uses in the Wild," *Sci. Robot.*, vol. 7, no. 66, eabm6074, 2022. doi: 10.1126/scirobotics.abm6074

[4] Z. Wei, S. Wang, K. Chen, and F. Wang, "ROS-Based Navigation and Obstacle Avoidance: A Study of Architectures, Methods, and Trends," *Sensors*, vol. 25, no. 14, 4306, 2025. doi: 10.3390/s25144306

[5] S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, and M. Ferguson, "From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2," *Robot. Auton. Syst.*, vol. 168, 104493, 2023. doi: 10.1016/j.robot.2023.104493

[6] D. Jo and Y. Kwon, "Generation of Critical Information and Sharing Mechanism for Multi-Robot Mission Success," *IEEE Access*, vol. 13, pp. 146855–146873, 2025. doi: 10.1109/ACCESS.2025.3600319

[7] F. Keramat, J. P. Queralta, and T. Westerlund, "Partition-Tolerant and Byzantine-Tolerant Decision Making for Distributed Robotic Systems with IOTA and ROS2," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12985–12998, 2023. doi: 10.1109/JIOT.2023.3257984

[8] S. Macenski, A. Soragna, M. Carroll, and Z. Ge, "Impact of ROS 2 Node Composition in Robotic Systems," *IEEE Robot. Autom. Lett.*, vol. 8, no. 7, pp. 3996–4003, 2023. doi: 10.1109/LRA.2023.3279614

[9] D. Cacciabue, J. Marino, F. Aglieco, M. Levorato, D. Perroni, and F. Risso, "Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge," in *Proc. 2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*, Saint Louis, 2024, pp. 49–54. doi: 10.1109/NetSoft60951.2024.10588914

[10] A. K. Kashyap and K. Konathalapalli, "Autonomous navigation of ROS2 based Turtlebot3 in static and dynamic environments using intelligent approach," *Int. J. Inf. Technol.*, pp. 1–23, 2025. doi: 10.1007/s41870-025-02500-5

[11] N. Pico, G. Mite, D. Morán, M. S. Alvarez-Alvarado, E. Auh, and H. Moon, "Web-Based Real-Time Alarm and Teleoperation System for Autonomous Navigation Failures Using ROS 1 and ROS 2," *Actuators*, vol. 14, no. 4, 164, 2025. doi: 10.3390/act14040164

[12] V. N. Oriakhi, "Autonomous Navigation for Differential Drive Robots: Grid-Based Fastslam with AMCL in ROS2," *Int. J. Latest Technol. Eng. Manag. Appl. Sci.*, vol. 14, no. 1, pp. 147–178, 2025. doi: 10.51583/IJLTEMAS.2025.1401016

[13] H. Liang, Y. Li, Q. Guo, and J. Yang, "ROS2-based locator optimized autonomous navigation robot," in *Proc. 2023 5th International Conference on Robotics, Intelligent Control and Artificial Intelligence (RICAI)*, Hangzhou, 2023, pp. 127–130. doi: 10.1109/RICAI60863.2023.10489438

[14] A. K. Nuhel *et al.*, "A ROS-2 based Path Planning and Maze Solving of Autonomous Robots," in *Proc. 2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS)*, Dhaka, 2024, pp. 1–6. doi: 10.1109/iCACCESS61735.2024.10499450

[15] S. M. Hussain, M. A. Atif, S. M. Daniyal, L. Ahmed, B. Memon, and Q. Pasta, "Navigating the Maze Environment in ROS-2: An Experimental Comparison of Global and Local Planners for Dynamic Trajectory Planning in Mobile Robots," in *Proc. 2024 International Conference on Robotics and Automation in Industry (ICRAI)*, Rawalpindi, 2024, pp. 1–7. doi: 10.1109/ICRAI62391.2024.10894622

[16] P. D. C. Cheng, M. Indri, F. Sibona, M. De Rose, and G. Prato, "Dynamic Path Planning of a mobile robot adopting a costmap layer approach in ROS2," in *Proc. 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Stuttgart, 2022, pp. 1–8. doi: 10.1109/ETFA52439.2022.9921458

[17] Y. Liu, Y. Lu, C. Peng, Q. Bu, Y. C. Liang, and J. Sun, "Autonomous Vehicle Based on ROS2 for Indoor Package Delivery," in *Proc. 2023 28th International Conference on Automation and Computing (ICAC)*, Birmingham, 2023, pp. 1–7. doi: 10.1109/ICAC57885.2023.10275227

[18] Y. Fang, H. Hafizh, and M. Ateeq, "An Autonomous Irrigation Robot Based on ROS2, YOLOv8 and Distributed ESP32 WMN Architecture," in *Proc. 2024 IEEE International Conference on Smart Internet of Things (SmartIoT)*, Shenzhen, 2024, pp. 384–391. doi: 10.1109/SmartIoT62235.2024.00065

[19] T. Schwörer, J. E. Schmidt, and D. Chrysostomou, "Nav2CAN: Achieving Context Aware Navigation in ROS2 Using Nav2 and RGB-D sensing," in *Proc. 2023 IEEE International Conference on Imaging Systems and Techniques (IST)*, Copenhagen, 2023, pp. 1–6. doi: 10.1109/IST59124.2023.10355731

[20] E. Sani, A. Sgorbissa, and S. Carpin, "Improving the ROS 2 Navigation Stack with Real-Time Local Costmap Updates for Agricultural Applications," in *Proc. 2024 IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, 2024, pp. 17701–17707. doi: 10.1109/ICRA57147.2024.10610984

[21] S. Macenski, F. Martin, R. White, and J. G. Clavero, "The Marathon 2: A Navigation System," in *Proc. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, 2020, pp. 2718–2725. doi: 10.1109/IROS45743.2020.9341207

[22] M. Szwedka and S. Budzan, "Improving the performance of Heuristic Path Planning Algorithms by a Computer Vision Obstacle Detection Method," in *Proc. 2025 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, Poznan, 2025, pp. 50–55. doi: 10.23919/SPA65537.2025.11215151

[23] V. Shcherbyna *et al.*, "Arena-Bench 2.0: A Comprehensive Benchmark of Social Navigation Approaches in Collaborative Environments," in *Proc. 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hangzhou, 2025, pp. 9202–9209. doi: 10.1109/IROS60139.2025.11246895

[24] Q. Serdel, C. Grand, J. Marzat, and J. Moras, "Online Localisation and Colored Mesh Reconstruction Architecture for 3D Visual Feedback in Robotic Exploration Missions," in *Proc. 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, 2022, pp. 8690–8697. doi: 10.1109/IROS47612.2022.9981137

[25] M. Ryalat, G. Al-Refai, N. Almtireen, and H. ElMoaqet, "Design of a ROS2-Based Hybrid Aerial-Ground Robot for Autonomous

Inspection Applications," *IEEE Access*, vol. 13, pp. 109274–109293, 2025. doi: 10.1109/ACCESS.2025.3582653

[26] B. Norbert, K. Gábor, and B. Áron, "Implementation of Trajectory Planning Algorithms for Track Serving Mobile Robot in ROS 2 Ecosystem," *Teh. Vjesn. - Tech. Gaz.*, vol. 30, no. 4, 2023. doi: 10.17559/TV-20220823131848

[27] N. Kumar, F. Fj, A. S. A. Doss, and N. G. M. Thao, "Hybrid Perception in Autonomous Mobile Robots: Integrating 2D and 3D SLAM with ROS2 for Enhanced Navigation and Mapping," in *Proc. 2025 1st International Conference on Consumer Technology (ICCT-Pacific)*, Matsue, 2025, pp. 1–4. doi: 10.1109/ICCT-Pacific63901.2025.11012875

[28] S. M. Elbouhy, A. Adamanov, P. M. Braun, and H. W. Rose, "Comparative Analysis of Local Trajectory Planning Algorithms in ROS2," in *Proc. 2025 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Palermo, 2025, pp. 1–6. doi: 10.1109/SIMPAR62925.2025.10979015

[29] T. Horelican, "Utilizability of Navigation2/ROS2 in Highly Automated and Distributed Multi-Robotic Systems for Industrial Facilities," *IFAC-Pap.*, vol. 55, no. 4, pp. 109–114, 2022. doi: 10.1016/j.ifacol.2022.06.018

[30] A. K. M J, A. V. Babu, S. Damodaran, R. K. James, M. Murshid, and T. S. Warrier, "ROS2 - Powered Autonomous Navigation for TurtleBot3: Integrating Nav2 Stack in Gazebo, RViz and Real-World Environments," in *Proc. 2024 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, KOTTAYAM, 2024, pp. 1–6. doi: 10.1109/SPICES62143.2024.10779642

[31] N. Pérez-Higueras, R. Otero, F. Caballero, and L. Merino, "HuNavSim: A ROS 2 Human Navigation Simulator for Benchmarking Human-Aware Robot Navigation," *IEEE Robot. Autom. Lett.*, vol. 8, no. 11, pp. 7130–7137, 2023. doi: 10.1109/LRA.2023.3316072

[32] B. Alsadik and S. Karam, "The Simultaneous Localization and Mapping (SLAM)-An Overview," *J. Appl. Sci. Technol. Trends*, vol. 2, no. 04, pp. 120–131, 2021. doi: 10.38094/sgej1027

[33] J. A. Placed *et al.*, "A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers," *IEEE Trans. Robot.*, vol. 39, no. 3, pp. 1686–1705, 2023. doi: 10.1109/TRO.2023.3248510

[34] T. Alhmiedat *et al.*, "A SLAM-Based Localization and Navigation System for Social Robots: The Pepper Robot Case," *Machines*, vol. 11, no. 2, 158, 2023. doi: 10.3390/machines11020158

[35] A. Dwijotomo, M. A. A. Rahman, M. H. M. Ariff, H. Zamzuri, and W. M. H. W. Azree, "Cartographer SLAM Method for Optimization with an Adaptive Multi-Distance Scan Scheduler," *Appl. Sci.*, vol. 10, no. 1, 347, 2020. doi: 10.3390/app10010347

[36] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *J. Open Source Softw.*, vol. 6, no. 61, 2783, 2021. doi: 10.21105/joss.02783

[37] R. Hanten, S. Buck, S. Otte, and A. Zell, "Vector-AMCL: Vector Based Adaptive Monte Carlo Localization for Indoor Maps," in *Advances in Intelligent Systems and Computing*, vol. 531, W. Chen, K. Hosoda, E. Menegatti, M. Shimizu, and H. Wang, Eds., Cham: Springer International Publishing, 2017, pp. 403–416. doi: 10.1007/978-3-319-48036-7_29

[38] I. Ullah, Y. Shen, X. Su, C. Esposito, and C. Choi, "A Localization Based on Unscented Kalman Filter and Particle Filter Localization Algorithms," *IEEE Access*, vol. 8, pp. 2233–2246, 2020. doi: 10.1109/ACCESS.2019.2961740

[39] M.-A. Chung and C.-W. Lin, "An Improved Localization of Mobile Robotic System Based on AMCL Algorithm," *IEEE Sens. J.*, vol. 22, no. 1, pp. 900–908, Jan.2022. doi: 10.1109/JSEN.2021.3126605

[40] Y. Li, L. Jiang, B. Tang, Y. Guo, B. Lei, and H. Liu, "Adaptive Monte Carlo Localization in Unstructured Environment via the Dimension Chain of Semantic Corners," *IEEE Trans. Ind. Inform.*, vol. 20, no. 7, pp. 9714–9724, 2024. doi: 10.1109/TII.2024.3385836

[41] R. Alqobali, R. Alnasser, A. Rashidi, M. Alshmrani, and T. Alhmiedat, "A Real-Time Semantic Map Production System for Indoor Robot Navigation," *Sensors*, vol. 24, no. 20, 6691, 2024. doi: 10.3390/s24206691

[42] C. Urrea and K. Valencia-Aragón, "Theoretical Analysis and Systematic Comparison of Local Navigation Control Strategies in Semi-Structured Environments: A Systems Approach," *Systems*, vol. 14, no. 3, 228, 2026. doi: 10.3390/systems14030228

[43] C. Moorthy, S. A, G. V, M. Manoj, V. Naveen, and P. H. Krishnan, "Comprehensive Exploration of Enhanced Navigation Efficiency via Deep Reinforcement Learning Techniques," in *Proc. 2024 Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI)*, Coimbatore, 2024, pp. 1087–1094. doi: 10.1109/ICoICI62503.2024.10696134

[44] C.-H. Cheng, K.-T. Huang, Y.-F. Huang, Y.-X. Xu, and K.-S. Liao, "Deep Learning Object Detection for Vehicle-to-Everything Systems with ROS 2.0 Distributed Communication Architecture," in *Proc. 2024 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*, Taichung, 2024, pp. 527–528. doi: 10.1109/ICCE-Taiwan62264.2024.10674437

[45] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *Proc. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, 2014, pp. 709–715. doi: 10.1109/IROS.2014.6942636

[46] J. Chae, H. Seo, S. Lee, and K.-J. Park, "Time-aware Costmap for Smoother and Less Disruptive AMR Navigation with ROS 2," *Int. J. Control Autom. Syst.*, vol. 23, no. 12, pp. 3587–3598, 2025. doi: 10.1007/s12555-025-0558-8

[47] N. K. Dhiman, D. Deodhare, and D. Khemani, "A ROS based framework for Multi-floor navigation for unmanned ground robots," in *Proc. the Advances in Robotics 2019*, Chennai, 2019, pp. 1–6. doi: 10.1145/3352593.3352654

[48] C. S. Chen, C. J. Lin, C. C. Lai, and S. Y. Lin, "Velocity Estimation and Cost Map Generation for Dynamic Obstacle Avoidance of ROS Based AMR," *Machines*, vol. 10, no. 7, 501, 2022. doi: 10.3390/machines10070501

[49] A. West, T. Wright, I. Tsitsimpelis, K. Groves, M. J. Joyce, and B. Lennox, "Real-Time Avoidance of Ionising Radiation Using Layered Costmaps for Mobile Robots," *Front. Robot. AI*, vol. 9, 862067, 2022. doi: 10.3389/frobt.2022.862067

[50] L.-M. Stan *et al.*, "Adaptive Costmap-based Path Planning in Partially Known Environments with Movable Obstacles," in *Proc. 2025 IEEE International Conference on Advanced Robotics (ICAR)*, San Juan, 2025, pp. 353–359. doi: 10.1109/ICAR65334.2025.11338633

[51] Y. Wang, B. Shen, L. Xiong, Z. Nan, and W. Tao, "Costmap A* Guided Reinforcement Learning Path Planning Method for Complex Environments Navigation," *J. Shanghai Jiaotong Univ. Sci.*, 2024. doi: 10.1007/s12204-024-2755-7

[52] L. Mao, G. Warnell, P. Stone, and J. Biswas, "PACER: Preference-conditioned All-terrain Costmap Generation," *IEEE Robot. Autom. Lett.*, vol. 10, no. 5, pp. 4572–4579, 2025. doi: 10.1109/LRA.2025.3549645

[53] J. F. Rascón, P. Reverté, X. Ruiz, M. S. Moura, D. Serrano, and C. Rizzo, "Leveraging Behavior Trees for Hybrid Autonomous Navigation in Seasonal Agricultural Environments," in *Proc. 2024 7th Iberian Robotics Conference (ROBOT)*, Madrid, 2024, pp. 1–7. doi: 10.1109/ROBOT61475.2024.10797423

[54] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior Trees and State Machines in Robotics Applications," *IEEE Trans. Softw. Eng.*, vol. 49, no. 9, pp. 4243–4267, 2023. doi: 10.1109/TSE.2023.3269081

[55] M. Moriconi, S. Laible, and C. Recchiuto, "Foundation-Model-Based Action Selection for Behavior Trees in Navigation," in *Proc. 2025 European Conference on Mobile Robots (ECMR)*, Padova, 2025, pp. 1–7. doi: 10.1109/ECMR65884.2025.11163061

[56] F. Storiale, E. Ferrentino, and P. Chiacchio, "Robot-Agnostic Interaction Controllers Based on ROS," *Appl. Sci.*, vol. 12, no. 8, 3949, 2022. doi: 10.3390/app12083949

[57] S. Zafar, N. Farooq-Khan, and M. Ahmed, "Requirements simulation for early validation using Behavior Trees and Datalog," *Inf. Softw. Technol.*, vol. 61, pp. 52–70, 2015. doi: 10.1016/j.infsof.2015.01.005

[58] H.-S. Park, S. Lee, D. Um, H. Ryu, and K.-J. Park, "An Analytical Latency Model of the Data Distribution Service in ROS 2," in *Proc. IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, London, 2025, pp. 1–10. doi: 10.1109/INFOCOM55648.2025.11044454

[59] M. Ruhe, K. Alba, M. Kipfmueller, and I. Mamaev, "Simulation-To-Reality Hyperparameter Optimization of MPPI Controllers via Bayesian Optimization in NVIDIA Omniverse Isaac Sim," in *Proc. 2025 IEEE 21st International Conference on Automation Science*

and Engineering (CASE), Los Angeles, 2025, pp. 874–880. doi: 10.1109/CASE58245.2025.11163854

[60] S. Salimpour, J. Peña-Queralta, D. Paez-Granados, J. Heikkonen, and T. Westerlund, "Sim-to-real reinforcement learning for local mobile robot navigation in dynamic environments," in *Proc. 2025 European Conference on Mobile Robots (ECMR)*, Padova, 2025, pp. 1–7. doi: 10.1109/ECMR65884.2025.11163286

[61] E. T. Hall, *The Hidden Dimension*, Garden City, N.Y.: Anchor Books, 1969.

[62] J. G. Clavero, F. M. Rico, F. J. Rodríguez-Lera, J. M. G. Hernandéz, and V. M. Olivera, "Impact of decision-making system in social navigation," *Multimed. Tools Appl.*, vol. 81, no. 3, pp. 3459–3481, 2022. doi: 10.1007/s11042-021-11454-2

[63] D. Mlinarček, R. Jánoš, M. Málik, J. Svetlík, J. Semjon, and Š. Ondočko, "Costmap Tuning for Autonomous Navigation: A Simulation and Real-World Study on the Hiwonder JetAcker," *Appl. Sci.*, vol. 16, no. 4, 1923, 2026. doi: 10.3390/app16041923