2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Tuning ROS 2 for Energy-Efficient Navigation: Empirical Insights from Costmap 2D Configurations

Michel Albonico*†, Andreas Wortmann‡, and Ivano Malavolta§
*Federal University of Technology - Paraná (UTFPR), DAINF-FB, Francisco Beltrão, Brazil
†University of Southern Denmark (SDU), MMMI, Centre for Software Technology (CST), Vejle, Denmark
‡Institute for Control Eng. of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Stuttgart, Germany
§Vrije Universiteit (VU) Amsterdam, Software and Sustainability (S2) Group, Amsterdam, The Netherlands
michelalbonico@utfpr.edu.br, mical@mmmi.sdu.dk, andreas.wortmann@isw.uni-stuttgart.de, i.malavolta@vu.nl

***Abstract*—Robots are increasingly used in diverse application areas, where autonomous navigation plays a central role. As these systems become more widespread, improving their energy efficiency is critical to extending operational time and reducing environmental impact. The Robot Operating System (ROS) is a widely adopted middleware for robotics, offering a rich set of configurable packages. However, this flexibility can result in suboptimal software configurations in dynamic environments, negatively affecting both performance and energy consumption. This paper investigates the impact of ROS 2 package reconfigurations on the energy efficiency of mobile robot navigation. We conduct a controlled experiment in two warehouse-like scenarios (small and large) with varying obstacle layouts and Costmap 2D configurations (essential to the Nav2 stack). Through repeated trials, we measure energy usage, power profile, CPU load, memory consumption, and navigation performance. Results show that configurations must be carefully chosen for the specific robotic environment, and we were able to identify critical settings that lead to good and poor performance and energy consumption.**

I. Introduction

Robots are increasingly deployed across diverse domains, *e.g.,* logistics, healthcare, agriculture, where energy efficiency is becoming an essential concern. As the demand for autonomous navigation grows, there is a need to reduce energy consumption, both for extending operational time (and consequently, increasing safety) and for minimizing environmental impact [1]. Among the various factors influencing robots' energy efficiency, software components play a central role, from low-level control to high-level decision-making [2], [3], [4]. In particular, robotics software can itself be a major source of energy consumption [5], making it a key factor in the design of sustainable and energy-aware robotic systems.

ROS has emerged as the de-facto standard for robotics software development [6], offering a wide range of highly configurable packages. This extensive configuration space, combined with dynamic robotic environments, can lead to suboptimal software configurations [7], which consequently affect both performance and energy efficiency [8], [9]. In this context, setting ROS configurations to specific workloads becomes a promising area for improvement [10].

The **main goal** of this study is to empirically assess the impact of ROS package (re-)configurations on the energy efficiency of mobile robots' navigation stack. We build our experiment on the Nav2[1], the current official navigation package for ROS 2, and propose alternative configurations for the *Costmap 2D* package[2], essential for navigation decision-making. This study hypotheses is that optimizing the *Costmap 2D* configuration in dynamic environments affects the robot's energy consumption during navigation tasks.

In this work, we conduct a controlled experiment in which a mobile robot autonomously navigates a warehouse floor populated with dynamic obstacles (i.e., objects deterministically added during the mission). The experiment has four independent variables: the *navigation goal*, the *Costmap 2D configuration*, the *obstacles* arrangement, and the *warehouse layout*. The *robot model*, *navigation goal*, and *other ROS package settings* are kept constant throughout the experiment. For each run, a fixed navigation goal is sent to the robot, a specific *Costmap 2D* configuration is applied, and the obstacle scenario is set deterministically. Each configuration was evaluated over 20x repetitions, balancing statistical reliability and experimental feasibility, which keeps the total execution time within practical limits. We collect performance and resource-usage metrics, including energy consumption.

Our **experimental results** demonstrate that tuning *Costmap 2D* parameters can reduce navigation time and power demands, leading to superior overall energy efficiency. Across both small and large warehouse layouts, certain parameterizations, notably those using a moderate resolution, low update frequencies, and reduced reliance on inflated or unknown-space layers, consistently achieved lower energy usage. In contrast, rough resolutions and excessive inflation values led to navigation failures, longer paths, or corrective behaviors, all of which increased energy consumption.

II. ROS Configuration

ROS packages tend to be highly configurable, given the dynamic environment where the robots are deployed [11]. That is, they expose a wide range of parameters, allowing users to customize system behavior without modifying source code. As a result, users can fine-tune performance for their specific hardware, environment, and application requirements. This flexibility, while powerful, introduces a significant challenge: the configuration space quickly becomes large and

[1]https://docs.nav2.org/
[2]https://docs.nav2.org/configuration/packages/configuring-costmaps.html

complex, which can lead to suboptimal configurations, compromising performance and resource usage. Formally, the ***ROS configuration space*** (C) of a system S can be defined as the Cartesian product of all valid parameter values across all ROS packages within S. The ***ROS system*** is defined as $S = \{\pi_1, \pi_2, \ldots, \pi_m\}$, where each π_j is a package. Each package contributes with a subset of parameters, where $P = \{p_1, p_2, \ldots, p_n\}$ is the complete set of parameters across all packages in S. For each parameter p_i, let D_i denote its *domain of admissible values*, which may be finite (e.g., $D_i = \{0, 1\}$ for booleans) or infinite (e.g., $D_i = \mathbb{R}$ for a continuous parameter value, or $D_i = [a, b]$ for an interval). The ***ROS configuration space*** is defined as $C = \prod_{i=1}^{n} D_i$. A *configuration* is then an element of this space, $c = (v_1, v_2, \ldots, v_n)$ with $v_i \in D_i$. Thus, each configuration $c \in C$ represents a complete assignment of admissible values to all parameters in the system S.

III. Related Work

Regarding the energy efficiency for Robot Operating System (ROS), Malavolta et al. [6] mined the ROS ecosystem to identify four green architectural tactics, achieving up to 7.9% energy savings on real hardware systems. Albonico et al. [12] presented a large-scale study on how the choice of programming language influences energy efficiency in ROS node implementations. Dordevic et al. [13] explored the energy implications of offloading ROS components to edge devices, providing empirical evidence on the trade-offs involved in distributed ROS deployments. Hammer et al. [9] evaluate the impact of different Simultaneous Localization and Mapping (SLAM) packages on the energy consumption of the ROS system. These studies form a foundation for our work, particularly regarding energy metrics and measurement methodologies. They diverge from our specific research focus and objectives since none of them focus on alternative configurations of Nav2 packages towards energy efficiency.

Considering robotic software (re-)configuration, Brugali [14] analyzes architectural impediments to runtime reconfiguration of ROS and proposes a reconfigurable navigation system, apart from the Nav2 stack. Peldszus et al. [15] investigate how robotic software can be customized, bringing empirical evidence on robotic software reconfiguration granularity and tools to support reconfiguration. Canelas et al. [16] systematically categorized 50 types of ROS misconfigurations from developer forum data, revealing that more than half remain unaddressed. Those works reinforce the importance of a broader empirical investigation regarding ROS configurations, and differ from our work since none of them measure the impact of robotic/ROS software configurations.

IV. Robotic Mission

This paper is based on a robotic mission in which a mobile robot navigates a warehouse floor, always going from its charging station to a fixed navigation goal. The mission represents a realistic situation in which the robot autonomously moves toward a shelf to pick an object. We model two warehouse floor layouts, which we simply call *maps* in the remainder of this document: a customized *small* map measuring $3 \times 4\,\text{m}$ and a publicly-available *large* one[3] ($\approx$ 10 times larger). Both maps contain different furniture, including storage racks, tables, and stacks of boxes that create navigation spaces of varying proportions (*i.e.*,, narrow and wide passages). The robot begins at a hypothetical charging station, with its navigation goal set at an opposite side of the warehouse, aiming at navigation paths of ≈ 6 (*small* map) and ≈ 18 (*large* map) meters, which triples the distance. The navigation in the *small* map is constrained in 180 seconds, while in the *large* map it is 540 seconds (also the triple). The environment may have obstacles added dynamically, deterministically and strategically placed so as not to completely block the robot's route, and in a time window that forces the robot to identify them after the navigation starts.

V. Study Design

By following the Goal Question Metric (GQM) framework [17], the **goal** of this study is defined as: ***analyze*** *the impact of* Costmap 2D *configurations* ***for the purpose of*** *improving the energy efficiency* ***with respect to*** *ROS 2-based mobile robotic systems* ***from the point of view*** *of researchers and practitioners*. The above-mentioned goal is achieved by answering four research questions:

RQ1: To what extent do configurations of *Costmap 2D* impact the energy consumption of ROS 2 system?

RQ2: Which specific configuration parameters of the *Costmap 2D* package contribute to energy efficiency during autonomous navigation?

RQ3: How do workload conditions relate to *Costmap 2D* configurations to influence energy consumption and system performance?

RQ4: How can the findings from this controlled experiment inform good practices for researchers and practitioners?

TABLE I: Metrics Used in the Study

Name	Unit	Description
Navigation Performance		
Time	seconds (s)	Duration for the robot to reach its goal.
Path Length	meters (m)	Total trajectory length from start to goal.
Success	count (#)	Trials in which the robot reaches the goal.
Recoveries	count (#)	Instances where recovery behaviors are triggered to handle failures.
Energy Usage		
Energy	joules (J)	Overall energy consumed during navigation.
Power	mWatts (mW)	Average power demand during navigation.
Resource Usage		
CPU	hertz (Hz)	CPU cycles spent by Nav2 processes.
Memory	kBytes (KB)	Peak memory allocated by Nav2 processes.

Table I summarizes the metrics used in the study. We consider three categories of **experimental variables**:

- *Static variables*: To ensure consistency and repeatability, certain aspects of the experimental setup are kept constant across all runs. These include the *robot model* (TurtleBot 4, representative of real-world vacuum robots), the *navigation*

[3]https://github.com/belal-ibrahim/dynamic_logistics_warehouse

stack (same Nav2 installation), the *hardware* and *virtual machines* (VMs), the *robot's initial position* (mimicking a charging station), and the *navigation goal*.

• *Independent variables*: The factors under investigation comprise (i) the *configurations* of the `Costmap 2D` package validated by two experts, (ii) the obstacle density (number of obstacles) within the environment, and (iii) the warehouse floor layout represented by two maps (small and large).

• *Dependent variables*: These correspond to the metrics previously introduced, grouped into *navigation performance*, *resource usage*, and *energy efficiency*.

VI. Experiment Execution

All experiment runs are conducted on a workstation with these specifications: Linux Ubuntu 22.04 (kernel `6.8.0-65-generic`), 16GB of RAM, a `12th Gen Intel® Core™ i7-12700F` processor with 20 cores (8 performance and 12 efficiency cores), and an NVIDIA `GeForce RTX 3060` GPU. Under this configuration, no system resource was saturated during experimentation. To guarantee isolation and reproducibility, we use four Docker (version `27.4.0`) containers: the `Gazebo` simulation environment, the `RViz` visualization tool, the `Nav2` navigation stack, and experiment orchestration services. For strict resource separation, both Docker and RViz containers were explicitly configured to run on the system's GPU. The Gazebo real-time factor (RTF) was fixed at 1.0, ensuring that the simulation advanced in synchrony with wall-clock time, thereby publishing sensor data at the same frequency as a physical robot. Within Gazebo, we adopted the standard *Turtlebot3 Waffle*[4] model distributed with ROS 2 Humble, which is natively equipped with a 2D LiDAR scanner. In addition, we extended the robot model by integrating a 3D depth camera using the public `realsense_gazebo_plugin`[5] package.

`Costmap 2D` Configurations – Table II presents the 18 Costmap 2D parameters considered in our experiment, which have been carefully selected based on ROS navigation tuning guides [18], [19]. These parameters and their possible values have been validated by two experts: (i) an academic researcher expert on Nav2 and (ii) the Nav2 project leader. An initial list of parameters and their suggested values were submitted to them via e-mail in June 2024 and refined based on their feedback. Although some parameters are continuous in nature, we restricted them to a finite set of representative values, which follows typical ranges based on default values, covering *low*, *medium*, and *high* values.

The full combination of all parameter values would result in more than 30 million configurations, making exhaustive experimentation impractical. We then use a pairwise combination strategy to generate a list of representative configurations for which experimentation is feasible. This technique, commonly employed in software testing [20], [21], is grounded in the observation that most software faults arise from interactions between pairs of parameters. Therefore, it should provide a representative coverage of the configuration space, possibly enabling the discovery of potential optimization opportunities. As a result, we generated 20 representative configurations to be used in the experiment (see details in the replication package[6]).

The pairwise configuration generation considers predefined constraints, mostly implicit parameters related to plugins, and specified conditions highlighted by the experts. The most relevant constraint is the combination of plugins proposed by the experts for three navigation strategies: i) *avoiding dynamic obstacles by an alternative path calculated globally*, ii) *avoiding dynamic obstacles by a global alternative path and local obstacle avoidance*, and iii) *avoiding dynamic obstacles only locally*. These combinations imply different trade-offs: from global replanning to hybrid global-local adaptation and local responsiveness. For this, four plugins are used: `static_layer` (`sl`), `obstacle_layer` (`ol`), `voxel_layer` (`vl`), and `inflation_layer` (`il`). The `sl` is responsible for representing immovable obstacles, which is always present in both local and global costmaps. The `ol` incorporates real-time 2D sensor data to detected obstacles. The `vl` models obstacles in three dimensions, relying on an RGB-D camera sensor. Finally, the `il` enlarges the perceived size of obstacles by adding safety margins around them.

Experiment orchestration – We use Robot Runner (RR) [22] to orchestrate the experiment, handling the setup and execution of both Gazebo and Nav2, and performance/energy profiling. RR sequentially prepares the environment, launches the simulation with the robot and its configuration, starts profilers to monitor resource, energy, and navigation performance, drives the robotic mission, and finally cleans up the environment to ensure isolation for subsequent runs. All the ROS 2 packages are set to use simulator time, which ensures that all nodes remain synchronized with the simulation clock instead of the system clock. After each run, a 60-second cooling-down period ensures that the machine returns to its previous resource usage levels. During the experiment, we track resource usage, energy consumption, and navigation performance using customized **profilers**. Energy consumption and CPU are measured via the RAPL interface through *PowerJoular* profiler [23]. We measure the power of the whole machine, and of the two main ROS 2 nodes linked to *Costmap 2D* configuration: `controller` and `planner`. Memory is measured via *psutil* Python library. The performance metrics are measured by customized profilers that collect Nav2 navigation feedback[7].

VII. Results

In this section, for a matter of space, we only illustrate the results of *small* map. However, all the data plotting is available on the replication package for further inspection. Experiments with the *small* map lasted a total of ≈ 33 h, with

[4]https://github.com/ROBOTIS-GIT/turtlebot3

[5]https://github.com/pal-robotics/realsense_gazebo_plugin

[6]https://github.com/IntelAgir-Research-Group/ROS2-EE-Reconf

[7]https://docs.nav2.org/plugin_tutorials/docs/writing_new_navigator_plugin

TABLE II: Costmap 2D parameter values used in the experiment

Parameter Name	Possible Values	Description
update_frequency	{1.0, 5.0, 10.0}	Rate (Hz) at which the costmap is updated.
resolution	{0.1, 0.2}	Resolution (in meters) of each cell in the costmap.
width	{1.0, 2.0, 3.0}	Width of the costmap in meters.
height	{1.0, 2.0, 3.0}	Height of the costmap in meters.
track_unknown_space	{true, false}	If true, unknown space is tracked and visualized.
transform_tolerance	{0.1, 0.5, 1.0}	Tolerance time for TF transform lookup.
plugin_combination	{ [sl, (sl,ol,il)], [(sl,ol), (sl,il)], [(sl, vl), (sl, ol, il)] }	Combination of layered costmap plugins used, divided in two sets (controller and planner plugins).
obstacle_max_range	{1.0, 2.0, 3.0}	Max range to consider obstacles from scan data.
raytrace_max_range	{1.0, 2.0, 3.0}	Max range for raytracing clears from scan data.
voxel_layer:z_voxels	{5, 10, 15}	Number of vertical voxels for 3D obstacle tracking.
voxel_layer:publish_voxel_map	{true, false}	Whether to publish the 3D voxel map.
voxel_layer:unknown_threshold	{5, 10, 15}	Threshold of unknown cells to consider a voxel unknown.
voxel_layer:mark_threshold	{0, 1, 2}	Number of marked voxels required to mark a grid cell.
inflation_layer:inflation_radius	{0.25, 0.5, 0.75}	Distance around obstacles to inflate.
inflation_layer:cost_scaling_factor	{1.0, 2.0, 3.0}	Rate at which inflation cost decreases with distance.
inflation_layer:inflate_unknown	{true, false}	Inflate unknown regions as obstacles.
inflation_layer:inflate_around_unknown	{true, false}	Inflate area around unknown space.
static_layer:subscribe_to_updates	{true, false}	If true, subscribes to external map updates.

the overall duration given by $T_{\text{total}}^{small} = (d + cd) \times c \times o \times r$, where d is the average duration of a single run, cd the cool down period, c the number of configurations, o the number of obstacle arrangements, and r the number of repetitions. Substituting the values, we obtain $T_{\text{total}}^{small} = (90 + 60) \times 20 \times 2 \times 20 \approx 120{,}000\,\text{s} \equiv 33{,}33\,\text{h}$. Experiments with *large* map lasted a total of 3 days ($T_{\text{total}}^{large} = (280 + 60) \times 20 \times 2 \times 20 \approx 272{,}000\text{s} \equiv 3.14\,\text{days}$).

A. Power and Energy Usage

Figure 1 presents the distribution of *power* and *energy* usage of the controller and planner nodes in the *small* map. For the controller node, all configurations consume less power than the *default* setup. In contrast, for the planner node, only c_2 and c_4 exhibit higher power consumption than the *default*. When considering aggregated *energy* ($E = navigation\_time \times (controller\_power + planner\_power)$, only three configurations c_2, c_4, and c_{12} achieve lower overall energy consumption. This reduction is mainly due to extended navigation times in the proposed configurations, which lower frequencies and promote more cautious obstacle avoidance, ultimately forcing the robot to move at reduced speeds. A notable case is c_4: even with high planner power, its short navigation time results in low overall energy consumption.

Figure 2 presents the distribution of CPU *power* (Figure 2a) and *energy* (Figure 2b) consumption of the entire machine used to run the experiments in the *small* map. Since the Gazebo and Rviz containers were executed on the GPU, they did not affect these CPU-based measurements. The results indicate that the configurations also directly affect the power and energy consumption of the entire Nav2 stack. The *power* of most configurations is lower than that of the *default* configuration, with the exception of c_9 and c_{13}. However, when considering *energy* consumption, only c_3 demonstrates higher efficiency, while c_4 and c_{12} achieve values close to the *default*.

B. Resource Usage

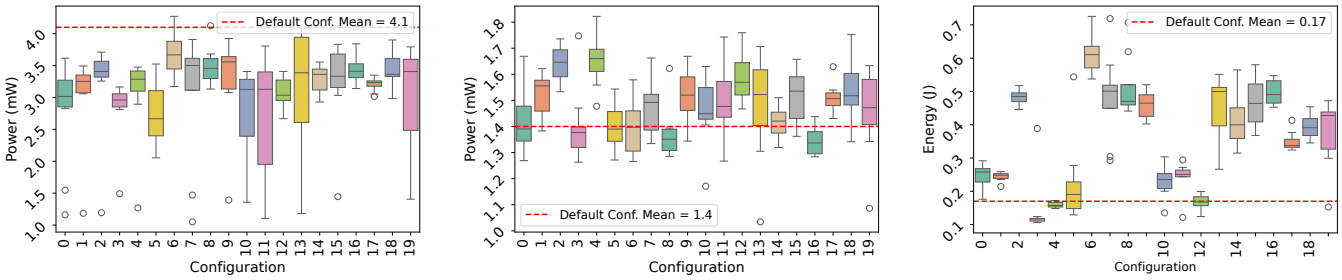
Figure 4 shows a variation in CPU usage across *Costmap 2D* configurations. This behavior is expected, as it reflects the corresponding power consumption of the machine. However, Figures 4a and 2a are not consistently aligned. This discrepancy arises because CPU usage is typically reported in terms of active cycles, which capture logical activity that may result in different power usages, given that modern processors employ complex architectural mechanisms, such as instruction-level parallelism, cache techniques, and dynamic voltage and frequency scaling (DVFS) [24].

Another observation is that memory usage also varies across *Costmap 2D* configurations, with differences even more pronounced than those observed for CPU usage. In particular, the memory usage of c_3, c_4, and c_{12} is consistently lower than that of the *default* configuration, corroborating the reduced energy consumption observed for the controller and planner nodes (Figure 1c). Since memory usage is not directly accounted for in our energy measurements, these results suggest that the impact of parameter configurations on the overall energy efficiency of the Nav2 stack may be even greater than indicated by CPU metrics alone.

C. Navigation Performance

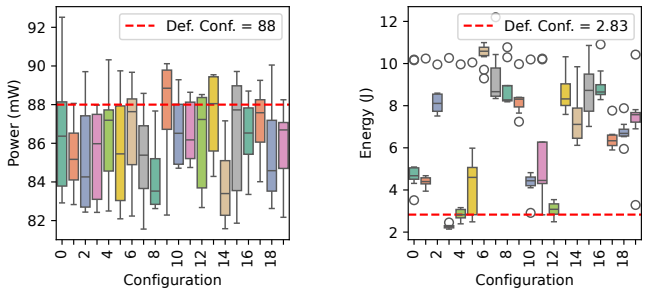
Figure 3a depicts **navigation time** across the 20 configurations in the *small* map. The reduced energy consumption observed in c_3, c_4, and c_{12} is primarily explained by their shorter navigation times. Several configurations (c_0, c_1, c_6, c_{10}, and c_{11}) result in intermediate navigation times of approximately $50\,\text{s}$, while the remaining configurations exceed $70\,\text{s}$.

Considering the calculated **path length**, the default configuration performs worse than most of the proposed alternatives, with a mean distance of 6.49 meters. Only c_1 results in a longer mean path, while c_2 produces a path of identical length. In contrast, most configurations yield paths at least one meter shorter, largely due to factors such as restricted inflation



(a) Power of `controller` node. (b) Power of `planner` node. (c) Energy consumption (both nodes).

Fig. 1: Power and energy consumption per configuration for `controller` and `planner` nodes in *small* map.



(a) Machine Power. (b) Machine Energy.

Fig. 2: Power and energy consumption per configuration for the whole machine in the *small* map.

layers around obstacles that encourage smoother or alternative trajectories. Path length does not seem to be determinant since c_9 more than doubles both navigation time and total energy consumption compared to the default, despite producing a path over one meter shorter, given the robot moves slower.

Figure 3c presents the percentage of **navigation success** for each configuration. Configurations c_6, c_8, c_{12}, c_{14}, c_{16}, c_{17}, and c_{18} achieve a 100% success rate across all 20 repetitions, indicating safe and reliable navigation. In contrast, c_5, c_{10}, c_{11}, c_{13}, and c_{19} proved problematic, with success rates below 80%. The causes of failures are analyzed in the discussion section. All remaining configurations completed more than 80% of the trials successfully; however, as we explain in the discussion, the few unsuccessful runs in these cases are attributable to unexpected conditions in experiment orchestration, with no flaws in the configurations themselves.

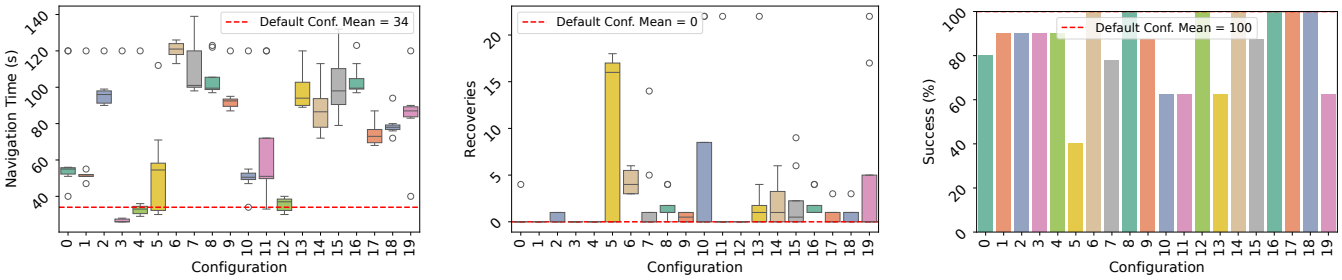
Figure 3b illustrates the number of **recoveries** across each configuration. All configurations with low success rates ($<$ 80%) also exhibit a large number of recoveries, reinforcing the interpretation that they are problematic, particularly c_5. Two distinct behaviors emerge when comparing c_6 and c_{11}. Configuration c_6 achieves successful navigation in all runs with a high number of recoveries, suggesting that it relies heavily on corrective actions to complete its tasks. In contrast, c_{11} reaches only a 60% success rate despite requiring no recoveries, indicating that its failures result from conditions that could not be mitigated by recovery strategies.

D. Large *Map Results*

A careful visual inspection of the *large* map plots leads to several relevant observations. Unexpectedly, although the map is $\approx 10\times$ larger than the *small* one, the mean CPU cycles, and consequently the measured power, are lower. This reduction is driven primarily by the `controller` node, whose power drops to roughly half, while the `planner` shows only a modest decrease. A plausible explanation is that the *large* map affords wider corridors, keeping the robot farther from walls and obstacles and therefore triggering fewer avoidance maneuvers and micro-corrections at the controller level. Because path planning is typically invoked less frequently than obstacle avoidance, both `planner` nodes exhibit similar power demands, where in the *large* map, the increased free space may also reduce path re-planning events.

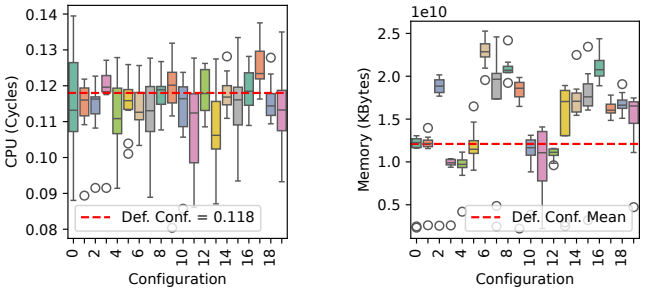
Memory usage also decreases, although the reduction is more modest. This is expected since the overall map is larger, while the `controller` node still processes the same map area per CPU cycle across configurations. We identify four configurations with discrepant memory usage: c_2, c_{17}, c_{18}, and c_{19}. All of them employ the `voxel` plugin, which alone does not appear to drive high memory usage, with elevated consumption emerging when it is combined with additional factors. Specifically, these configurations also apply broader *inflation* of obstacles and enable *tracking of unknown space*, both of which expand the costmap representation. Interestingly, under the *small* map setting, c_2 stands out as one of the most memory-efficient configurations. Its main difference lies in the `cost_scaling_factor`, which is set to 3 in c_2, whereas in the other configurations it is fixed at 1. This higher scaling factor makes the cost of distant obstacles decrease faster, so the inflated area is narrower, which reduces the number of cells stored and updated, contributing to lower memory usage. It's particularly low memory usage on the *small* map is also likely influenced by the limited extent of the map.

The navigation path length approximately triples in the *large* map, which is also the case for the *navigation time*. This means the robot moves at a similar pace regardless of the map layout. We also observe an unexpected behavior associated with excessive values of the *inflation layer*. Such settings induced the robot to favor trajectories in close proximity to obstacles rather than exploiting the available free space. A



(a) Navigation time. (b) Navigation recoveries. (c) Navigation success.

Fig. 3: Navigation performance per `Costmap_2D` configuration in the *small* map.



(a) Machine CPU cycles. (b) Machine memory usage.

Fig. 4: CPU and memory usage per configuration for the whole machine in the *small* map.

representative case is c_8, where the navigation path was recalculated upon the introduction of the first obstacle. Instead of increasing clearance, the robot consistently detoured through narrow passages, for instance, between an obstacle and a pile of boxes. This behavior resulted in unnecessarily long and oscillatory trajectories, reflecting the tendency of the robot to remain near object boundaries.

E. Statistical Analysis

We conducted a series of statistical tests for both the *small* and *large* maps. For each map, we examined the effect of (i) configuration and (ii) number of obstacles on the measured power of the `controller` and `planner` nodes, as well as on overall energy consumption. We also (iii) statistically compared the measurements for the two maps. Shapiro–Wilk test [25], in most cases, rejects the null hypothesis of normality, particularly for the *small map* experiments, indicating non-normal distributions. Given the frequent violations of normality, non-parametric tests were selected.

1) Effects of configurations: For the *small map*, the test showed a strong and statistically significant effect of configuration on the `controller` power (H=222.469, p=1.102×10^{-36}), the `planner` power (H=237.752, p=9.255×10^{-40}), and the overall energy usage (H=222.469, p=1.102×10^{-36}). For the *large map*, configurations also had a significant influence, with smaller effect sizes: `controller` (H=188.450, p=6.652×10^{-30}), `planner` (H=138.150, p=4.118×10^{-20}), and overall energy (H=188.450, p=6.652×10^{-30}). Post-hoc tests revealed numerous significant pairwise differences among configurations in both maps, with specific parameterizations outperforming others.

2) Effects of Obstacles: Across both maps, obstacles did not yield statistically significant effects. In the *small map*, Kruskal–Wallis results were non-significant for the `controller` (H=1.479, p=0.224), `planner` (H=0.026, p=0.871), and overall energy (H=1.479, p=0.224). Similarly, in the *large map*, results were non-significant for the `controller` (H=1.720, p=0.190), `planner` (H=1.741, p=0.187), and overall energy (H=1.720, p=0.190). This indicates that, within the tested ranges, obstacle density had a limited impact compared to configuration choices.

3) Cross-Run Comparisons (Small vs. Large Map): Direct comparisons between maps showed marked differences. For the `controller`, Kruskal–Wallis indicated a very strong difference between maps (H=683.666, p=1.066×10^{-150}), mirrored in overall energy (H=683.666, p=1.066×10^{-150}). The `planner` also differed across maps, albeit to a lesser extent (H=25.863, p=3.666×10^{-7}). A two-way ANOVA (*map* $\times$ *obstacles*) corroborated these findings: a significant main effect of *map* for `controller` (F=690.28, p=2.76×10^{-126}), `planner` (F=33.52, p=8.53×10^{-9}), and overall energy (F=690.28, p=2.76×10^{-126}), with no significant main effect of *obstacles* (all p>0.17) and no significant interaction (all p>0.27). Together with the post-hoc results, this confirms that map size/structure and configuration dominate energy/performance outcomes, while obstacle density does not modulate these effects in our setting.

VIII. Discussion

A. Answering the Research Questions

A–RQ1: Our experiments confirm that *Costmap 2D* configurations have a substantial impact on the energy consumption of the Nav2 stack and, consequently, on the overall energy efficiency of ROS 2 systems. Parameter variations affect both the power profile of the planner and controller nodes as well as navigation time, which together determine total energy usage. Configurations such as c_3, c_4, and c_{12} consistently achieved lower energy consumption than the default setup, primarily by reducing navigation time and lowering node-level power. These configurations also exhibited reduced memory usage, an aspect not directly captured in our power and energy measurements, which may represent an additional dimension

of energy efficiency improvement. Overall, these findings demonstrate that *the energy efficiency of the Nav2 stack can be improved through deliberate configuration tuning*.

A–RQ2: The analysis reveals that parameter tuning has a greater influence than plugin combinations. Three parameters stand out: *resolution* – a moderate value ($0.1\,m$) reduced computational load without compromising navigation reliability, while a rough setting ($0.2\,m$) consistently caused unsuccessful navigation; *update frequency* – higher frequencies ($\geq 5\,Hz$) improved responsiveness and lowered `controller` power, whereas very low values ($1\,Hz$) increased `planner` power due to more intensive replanning; *inflation* and *unknown-space handling* – disabling `inflate_around_unknown` and `track_unknown_space` parameters reduced unnecessary complexity in costmaps, shortening navigation times.

A–RQ3: Map size and layout are directly related to how *Costmap 2D* configurations affect energy consumption and Nav2 performance. In the *large* map, although the environment is $\approx 10\times$ larger, the mean CPU cycles and power are lower than in the *small* map. This is mainly linked to the `controller` node, which in wider corridors reduces the micro-corrections in the robot's path and faces farther objects. Contrastingly, narrow environments force frequent avoidance and re-planning, making configuration choices more critical: combinations of `voxel` layers, broad obstacle inflation, and `track_unknown_space` increase memory usage, while higher `cost_scaling_factor` narrows inflated regions, leading to improved efficiency. Thus, *performance-energy trade-offs must also consider different environmental conditions when setting the Nav2 parameterization*. The obstacle arrangements used in our experiments did not produce statistically significant differences in the measurements, even though they were deterministically placed to trigger path re-planning. Nevertheless, experiments involving a larger number of obstacles, arranged in more dynamic ways, could affect the measurements differently, requiring further experimentation.

A–RQ4: The findings suggest practical guidelines: i) parameter tuning should be prioritized over plugin selection for energy efficiency; ii) configurations must be validated under realistic workloads, as settings that appear efficient in simple environments may fail in more complex and large maps; iii) navigation time should be closely monitored, as it is the most influential variable for energy consumption; and iv) results underscore the need to integrate energy measurement into the ROS development workflow, enabling practitioners to detect inefficient parameter settings early and researchers to build systematic knowledge for energy-aware robotics software.

B. Practical Observations

Parameter tuning matters more than the combination of plugins – Analyzing the configurations that yield lower power and energy consumption, we find no consistent pattern in the choice of local and global plugins, suggesting they are not decisive factors. This observation is particularly worthy of note, as certain plugins, such as the `voxel_layer`, are expected to be computationally demanding due to three-dimensional representations of the environment.

The configurations that outperform the *default* in terms of energy efficiency (c_3, c_4, and c_{12}) share three characteristics: shorter *navigation times*, lower power usage in the `planner` and/or `controller` nodes, and the absence of recovery behaviors. These configurations also follow a common parameter pattern: `resolution` set to 0.1, `inflate_around_unknown` disabled, and a high `update_frequency` (≥ 5). In contrast, the *default* configuration uses a finer `resolution` of 0.05, which emerges as the main differentiating factor. Lower `resolution` reduces the number of grid cells, simplifying the costmap and lowering the computational load, which in turn decreases power consumption. However, setting the `resolution` too high (0.2) leads to a large number of unsuccessful navigation attempts, highlighting the need for careful calibration of this parameter.

Navigation time is variable and determinant – In the graphs of Figure 1, navigation time emerges as one of the most variable dependent measures. This variable is critical for estimating the total energy consumption of a robotic mission, as longer operating times inherently lead to greater energy usage. Among the configurations with the longest navigation times (c_2, c_6, c_7, c_8, and c_{16}), a consistent pattern can be observed: `inflation_radius` is set to `0.5`, and `track_unknown_space` is enabled. A large inflation radius (`0.5`) reduces available free space by enlarging obstacles, forcing detours, while enabling `track_unknown_space` further restricts motion by treating unexplored areas as unsafe. In the *large* map, it was observed that a large inflation radius forces the robot to navigate close to obstacles, avoiding open spaces, which increases the navigated distance and time. For the configuration with the shortest navigation time (c_3), `inflation_radius` is set with value 0.35 and `track_unknown_space` is disabled, confirming them as determining factors in reducing navigation time. However, since those parameters are equally set in configurations c_9, which exhibit longer navigation time. The main difference between c_3 and c_9 is the `transform_tolerance`=0.1 (vs. 0.5), making the costmap wait longer for buffered transforms in c_9, accumulating pauses during navigation. So, navigation time seems to result from a combination of those factors.

Reasons behind unsuccessful navigation – An analysis of the configurations leading to unsuccessful navigation reveals that some configurations performed particularly poorly ($< 80\%$), with failures primarily due to crashes where the robot collided with obstacles, especially at sharp corners. These crashes are likely explained by a combination of `Costmap 2D` *resolutions*, *restrictive sensing ranges*, and *limited local planning space*. With a resolution of 0.2 m, the local costmap lacks sufficient granularity to represent narrow passages or fine environmental details, causing the robot to misinterpret free space and obstacle boundaries [18]. Furthermore, restricting obstacle detection to only 1 m results in late obstacle recognition, reducing the system's reaction time. Finally, the absence of a local inflation layer eliminates

smooth navigation gradients, increasing the likelihood of the robot hitting obstacles.

For configurations with a success rate above 80% that still resulted in at least one unsuccessful navigation, we did not observe any crashes. Upon closer inspection of the experimental data, we found that these runs terminated after only 20 seconds, with no navigation performance collected. This deterministic behavior occurs when the robot fails to receive the navigation goal within 15 seconds, a situation that may arise from transient or unstable conditions in the experimental environment rather than from the navigation process itself. Since these cases are artifacts of the experimental setup and do not reflect the robot's actual navigation capability, they can be safely treated as outliers. These outliers represent less than 10% of the collected runs, and their removal does not distort the underlying data assumptions.

Configurations are not straightforward – Configuring the *Costmap 2D* parameters in Nav2 is not straightforward when the goal is to balance performance and energy consumption. Our results show that the *default* configuration is not necessarily ideal, even for simple missions, as it reflects general-purpose trade-offs rather than mission-specific optimizations. Moreover, performance and energy efficiency emerge from the interplay of multiple parameters rather than from any single setting, which becomes even more critical in larger or dynamic environments where replanning and obstacle avoidance are more frequent. Therefore, there is a need for adaptive configuration strategies: while static tuning is useful for controlled experiments, real deployments require mechanisms that reconfigure parameters at runtime.

Threats to external validity – Our experimental evaluation is limited to the Nav2 navigation stack, specifically focusing on the *Costmap 2D* package. Moreover, all experiments were conducted using a single robot model, with fixed sensors and hardware settings. Future work will replicate the experiments across diverse robot platforms and extend the analysis to additional ROS 2 packages.

IX. Conclusion

This paper empirically demonstrates that ROS 2 *Costmap 2D* configurations significantly influence the energy efficiency of mobile robot navigation. Through controlled experiments across different maps and obstacle layouts, we showed that parameter tuning, particularly *resolution*, *update frequency*, and *inflation settings*, has a stronger impact on energy consumption than plugin combinations. Configurations such as c_3, c_4, and c_{12} consistently reduced *navigation time*, lowered *power* demands of `controller` and `planner` nodes, and decreased *memory* usage, thus outperforming the *default* setup. Our findings highlight that **configuration tuning is essential to achieve efficient navigation in ROS 2 systems**.

References

[1] E. G. Hertwich and R. Wood, "The growing importance of scope 3 greenhouse gas emissions from industry," *Environmental Research Letters*, 2018.

[2] K. Chinnappan, I. Malavolta, G. A. Lewis, M. Albonico, and P. Lago, "Architectural tactics for energy-aware robotics software: A preliminary study," in *ECSA*, 2021.

[3] M. Albonico, I. Malavolta, G. Pinto, E. Guzman, K. Chinnappan, and P. Lago, "Mining energy-related practices in robotics software," in *IEEE/ACM MSR*, 2021.

[4] M. G. dos Santos, B. M. Napoleão, F. Petrillo, D. Ameyed, and F. Jaafar, "A preliminary systematic mapping on software engineering for robotic systems: A software quality perspective," in *IEEE/ACM ICSEW*. ACM, 2020.

[5] S. Swanborn and I. Malavolta, "Energy efficiency in robotics software: A systematic literature review," in *IEEE/ACM ASE*, 2020.

[6] I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago, and D. Garlan, "Mining guidelines for architecting robotics software," *JSS*, 2021.

[7] J. A. Pereira, M. Acher, H. Martin, J.-M. Jézéquel, G. Botterweck, and A. Ventresque, "Learning software configuration spaces: A systematic literature review," *Journal of Systems and Software*, 2021.

[8] M. Dordevic, M. Albonico, I. Malavolta, G. Lewis, and P. Lago, "Computation offloading for ground robotic systems communicating over wifi – an empirical exploration on performance and energy trade-offs," *ESEM*, 2023.

[9] E. Hamer, M. Albonico, and I. Malavolta, "Resource utilization of 2d slam algorithms in ros-based systems: an empirical evaluation," *Journal of the Brazilian Computer Society*, 2025.

[10] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, 2019.

[11] P. Canelas, B. Schmerl, A. Fonseca, and C. S. Timperley, "Understanding misconfigurations in ros: An empirical study and current approaches," in *ACM ISSTA*, 2024.

[12] M. Albonico, M. B. Canizza, and A. Wortmann, "Energy efficiency in ros communication: A comparison across programming languages and workloads," *Frontiers in Robotics and AI*, 2025.

[13] Milica Dordevic and Michel Albonico and Grace Lewis and Ivano Malavolta and Patricia Lago, "Computation offloading for ground robotic systems communicating over wifi–an empirical exploration on performance and energy trade-offs," *ESEM*, 2023.

[14] D. Brugali, "Runtime reconfiguration of robot control systems: a ros-based case study," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, 2020.

[15] S. Peldszus, D. Brugali, D. Strüber, P. Pelliccione, and T. Berger, "Software reconfiguration in robotics," *Empirical Software Engineering*, 2025.

[16] P. Canelas, B. Schmerl, A. Fonseca, and C. S. Timperley, "Understanding misconfigurations in ros: An empirical study and current approaches," in *ISSTA*, 2024.

[17] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (GQM) approach," *Encyclopedia of software engineering*, 2002.

[18] K. Zheng, *ROS Navigation Tuning Guide*. Springer, 2021.

[19] "ROS 2 Navigation Tuning Guide – Nav2," 2024. [Online]. Available: https://automaticaddison.com/ros-2-navigation-tuning-guide-nav2/

[20] M. Albonico, S. D. Alesio, J.-M. Mottu, S. Sen, and G. Sunyé, "Generating test sequences to assess the performance of elastic cloud-based systems," in *IEEE CLOUD*, 2017.

[21] K.-C. Tai and Y. Lei, "A test generation strategy for pairwise testing," *IEEE transactions on Software Engineering*, 2002.

[22] S. Swanborn and I. Malavolta, "Robot runner: A tool for automatically executing experiments on robotics software," in *Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering*. ACM, 2021, pp. 33–36.

[23] A. Noureddine, "Powerjoular and joularjx: Multi-platform software power monitoring tools," in *IE2022*, 2022.

[24] J.-P. Halimi, B. Pradelle, A. Guermouche, N. Triquenaux, A. Laurent, J. C. Beyler, and W. Jalby, "Reactive dvfs control for multicore processors," in *IEEE GreenCom and iThings and IEEE CPSS*, 2013.

[25] S. S. Shapiro, M. B. Wilk, and H. J. Chen, "A comparative study of various tests for normality," *Journal of the American statistical association*, 1968.