


Resource Utilization of 2D SLAM Algorithms in ROS-Based Systems: an Empirical Evaluation

Engel Hamer  [Vrije Universiteit Amsterdam | e.hamer@student.vu.nl]

Michel Albonico  [Federal University of Technology | michelalbonico@utfpr.edu.br]

Ivano Malavolta   [Vrije Universiteit Amsterdam | i.malavolta@vu.nl]

 Department of Informatics, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Received: 03 April 2024 • Accepted: 06 February 2025 • Published: 10 April 2025

Abstract Simultaneous localization and mapping (SLAM) is an important task in robotic systems, which entails mapping an environment while keeping track of the robot's position within the created map. The Robot Operating System (ROS) offers various packages for this functionality, where each one may lead to different performance and resource usage. Therefore, this study aims to investigate the impact of different ROS-based SLAM algorithms on resource utilization, including possible trade-offs with performance (e.g., the accuracy of the created map). The investigation is centered on primary experiments involving multiple runs of a single robot, which alternates between four SLAM algorithms: Cartographer, Gmapping, Hector SLAM, and Karto. During these experiments, the robot autonomously navigates through two types of arenas: point-to-point (multi-goal navigation) and circular (returning to the starting position after following the perimeter). Throughout these trials, the robot's performance is assessed based on the ROS system's efficiency and energy consumption. In a secondary set of experiments, the tests are repeated, but with key SLAM algorithm parameters reconfigured to evaluate their impact. The experiment results reveal Karto as the most efficient algorithm across all evaluated metrics, and the one that creates the most visually consistent maps. Cartographer was the algorithm that showed the least promising results regarding both, energy consumption and CPU utilization. Furthermore, Gmapping was the algorithm most susceptible to changes in SLAM algorithms' parameter values. The results presented in this study are, therefore, key for resource-aware design choices when using SLAM algorithms in the context of ROS-based systems.

Keywords: ROS, Software Engineering, Empirical Evaluation, Energy Consumption, SLAM

1 Introduction

As the scope and complexity of robotic systems continue to grow, their development requires increasingly deep expertise in diverse domains [Estefo *et al.*, 2019; Kolak *et al.*, 2020]. To overcome this challenge, initiatives to encourage software reuse and collaboration among the robotics community have been pivotal in the success of the field. The *Robot Operating System* (ROS) [Quigley *et al.*, 2009] is the most popular of these, often considered as the de-facto standard in robotic systems development in both research and industry [Malavolta *et al.*, 2020; Kolak *et al.*, 2020]. ROS offers communication middleware that is vital to most common robotic systems today, as well as an expanding collection of *packages* that provide modular functionality for a wide variety of tasks a robotic system may require.

An important software component of many robotic systems is *simultaneous localization and mapping* (SLAM), where a robot maps the environment while localizing itself in the create map [Smith and Cheeseman, 1986; Stachniss *et al.*, 2016]. The obtained map can be used by other system tasks (e.g., path planning or visualization of the environment to a human operator) [Cadena *et al.*, 2016]. Example applications of SLAM range from consumer systems, such as autonomous vacuum cleaners, to industrial warehouse robots. Given its widespread use, a multitude of packages offering SLAM functionality is available in ROS.

In the domain of mobile robots, energy is a finite resource that directly affects the maximum operational time: a full battery charge will last shorter if the robot's power consumption is higher. Furthermore, hardware in such systems may have limited computational power. Considering that SLAM applications inherently use mobile robots, the *resource utilization* that SLAM algorithms impose on a robotic system is thus highly relevant. On example are robots for *urban search and rescue* (USAR), where SLAM is used to map a disaster area [Kleiner and Dornhege, 2007] and to identify potential victims [Balaguer *et al.*, 2009]. In such situations where time is critical and resources are sparse, it is desirable for a SLAM algorithm to cause as little drain of the robot's battery as possible. Simultaneously, sufficient *accuracy* of the created map is required to prevent humans from unnecessarily exposing themselves to dangerous situations.

This study aims to investigate the impact of SLAM algorithms on *resource utilization* in the context of ROS-based systems, including possible trade-offs between resource consumption and *accuracy* of the created map. It achieves this goal by answering two main research questions:

- **RQ1:** *How do SLAM algorithms impact resource utilization of ROS-based systems?*
- **RQ2:** *What are the trade-offs between resource consumption and the accuracy of generated maps for SLAM algorithms in ROS-based systems?*

The target audience of this study consists of researchers and practitioners working with ROS-based robotic systems. It aids *researchers* by (i) providing insight into how various SLAM paradigms impact resource utilization, (ii) identifying which paradigms are preferable when designing resource-aware SLAM algorithms, and (iii) offering a timestamped dataset of energy, CPU, and memory measurements along with rosbag recordings and system logs over a total of 300 SLAM runs in a physical robot. This study also aids *practitioners* in choosing which of the SLAM algorithms available in the ROS ecosystem to use for their application. There exists no single best solution to the SLAM problem, but instead, the practitioner's choice depends on factors such as the desired map resolution, update frequency, and the nature of the features in the map [Stachniss *et al.*, 2016]. By providing recommendations regarding trade-offs between resource utilization and the accuracy of the SLAM algorithms, this study bases resource-aware design choices when architecting robotic systems.

The main *contributions* of this study are: *i) design and usage recommendations* for resource-aware SLAM algorithms in the context of ROS-based systems, with a focus on the trade-off between energy consumption and accuracy; *ii) an empirically obtained datasets* containing measurements of power consumption, CPU utilization and memory utilization along with rosbag recordings and system logs over a total of 300 SLAM runs with a physical robot (TurtleBot3 Burger); *iii) ROS profiling packages* designed to collect CPU/memory and power consumption measurements from physical robots with minimal overhead, valuable for ROS 1 and ROS 2; *iv) a framework for evaluating SLAM algorithms* implemented in ROS packages, which consists of ROS *launch* files that can be adapted to other algorithms than the ones evaluated in this study, and a configuration file for execution using *Robot Runner* [Swanborn and Malavolta, 2021]. *v) a replication package* containing all scripts for the experiment execution and analysis. This enables the independent verification and replication of this study.

The remainder of the paper is organized as follows: Section 2 introduces the main concept of ROS, and SLAM and its solutions. Section 3 positions this study in the current research field by comparing it to related work. Section 4 presents the experiment definition based on the Goal Question Metric (GQM) framework. Section 5 defines the choice of subjects and variables in this study, and motivates each of them, as well as describes the approach used for analyzing the data obtained from the experiment. Section 6 illustrates how the experiment is set up and describes the steps performed during actual execution. Section 7 presents the obtained data through plots and statistical analysis. Section 8 provides an interpretation of the obtained results and the recommendations for the resource-aware design and usage of SLAM algorithms in ROS. Section 9 categorizes several threats to validity that apply to this study, and the measures taken to mitigate them. Finally, Section 10 concludes the paper by summarizing the key contributions and suggesting directions for future research.

2 Background

This section presents some fundamental concepts related to SLAM and ROS which may not be trivial to the reader. These concepts are vital to the design of the experiment and the subsequent interpretation of its results.

2.1 The Robot Operating System

ROS is the most popular robotics middleware in both industry and research [Malavolta *et al.*, 2020; Kolak *et al.*, 2020]. Unlike the name suggests, ROS is not an OS in the colloquial sense of the term but instead provides a standardized communication layer on top of a host operating system [Quigley *et al.*, 2009]. Furthermore, the ROS ecosystem consists of 6527 packages available at the time of writing¹ and an active community of researchers, practitioners, and robotics enthusiasts.

ROS-based systems comprise loosely coupled *nodes*: processes that perform computations and communicate with each other through ROS *messages*. Nodes can be thought of as modules, which can offer any kind of functionality. In the context of SLAM, common system configurations include one node running the actual SLAM algorithm and another that uses the obtained map, e.g., path planning.

Message exchange takes place using a one-way communication model, where nodes can publish or subscribe to *topics*. Depending on the data in which a node is interested, it will subscribe to the topics that are relevant to it. In addition to this *publish-subscribe* model, ROS also offers client-server interaction through *services*.

2.2 Simultaneous Localization and Mapping

The notion of SLAM as introduced in Section 1 can be elaborated through (i) a problem definition, (ii) known solution paradigms, and (iii) the sensory hardware as input for solving the SLAM problem.

2.2.1 Problem definition

Concretely, the problem that SLAM entails in modern robotics is defined by Stachniss *et al.* [2016]; Durrant-Whyte and Bailey [2006] as follows:

"A mobile robot roams an unknown environment, starting at an [unknown] initial location. Its motion is uncertain, making it gradually more difficult to determine its current pose in global coordinates. As it roams, the robot can sense its environment with a noisy sensor. The SLAM problem is the problem of [incrementally] building a map of the environment while simultaneously determining the robot's position relative to this map given noisy data."

While the definition of SLAM is easy, the notion of *noise* is the fundamental reason that makes the problem itself difficult to solve. This is best understood by considering SLAM

¹<https://index.ros.org/stats/>

as a probabilistic formula [Stachniss *et al.*, 2016]:

$$p(x_t, m \mid Z_t, U_t) \quad (1)$$

Where:

- x_t denotes the robot *pose* at time t ;
- m denotes the *map* (or model) of the environment;
- $Z_t = \{z_1, z_2, \dots, z_t\}$ denotes the sequence of measurements obtained from the robot's sensors from the start of the execution until time t ;
- $U_t = \{u_1, u_2, \dots, u_t\}$ denotes the sequence of movement instructions sent to the robot from the start of the execution until time t .

If U_t were noise-free, then the robot's pose relative to the starting point (x_t) could be derived from this sequence alone. The change in pose would follow directly from the accumulated movement controls. Consequently, if Z_t were noise-free, the task of aligning measurements to form a map of the environment (m) with absolute accuracy would be a trivial task. In practice, these sensors and instructions do suffer from noise. The robot cannot measure x_t and m directly, so solving Equation (1) therefore intuitively comes down to interpreting and aggregating the robot's noisy sensor readings with its acquired spatial knowledge of the environment in an optimal way.

2.2.2 Solution paradigms

Over the past three decades, SLAM has been actively studied in the robotics domain [Cadena *et al.*, 2016]. During this time, three main solution paradigms have emerged, being (i) extended Kalman filters, (ii) particle filters, and (iii) graph-based methods [Stachniss *et al.*, 2016]. One important aspect that all paradigms have in common is their reliance on probabilistic techniques for solving parts of the SLAM problem, which dominate the research field as the sole solution for solving SLAM [Thrun *et al.*, 2002].

The earliest algorithms aimed at solving the SLAM problem store an estimate of the robot's location and all environmental features in a single state vector. Uncertainties among these estimates are represented by an accompanying error covariance matrix. The *extended Kalman filter* (EKF) is then used to update both the state vector and covariance matrix as the robot traverses the environment [Cheeseman *et al.*, 1987]. Any newly observed features are added to the state vector, resulting in a quadratic growth of the covariance matrix [Stachniss *et al.*, 2016]. This exposes an obvious scaling problem for EKF-SLAM approaches.

With the proposal of their *FastSLAM* algorithm [Montemerlo *et al.*, 2002], Montemerlo *et al.* [2002] laid the foundation for a next paradigm in SLAM algorithms by exploiting *particle filters* (PF). These rely on sampling several possible robot poses (particles) based on the movement model, and then subsequently interpreting their distribution to approximate the probability distribution of the true state.

The most recent of the three paradigms in SLAM follows a *graph-based* approach. Environmental features and robot locations are represented as nodes in a graph, with edges between nodes representing the robot location in subsequent

time steps. These edges are based on U_t as in Equation (1). Additional edges associate environmental features with a robot location x_t if the sensory readings contained the feature at time t . Interpreting all edges as constraints, the estimated map and full trajectory of the robot are then obtained by relaxing constraints according to sparse linear optimization techniques [Stachniss *et al.*, 2016].

To varying degrees, the methods described here have proven to be robust and useful for mapping static, structured environments of limited size. Challenges that come with unstructured, dynamic, and large-scale environments are open challenges in current research on SLAM [Stachniss *et al.*, 2016; Cadena *et al.*, 2016].

2.2.3 Sensors

Modern robotic systems are equipped with a wide variety of components for sensing their environment. Among the most common sensor technologies used for SLAM are *laser range finders*, which rely on laser pulsing to determine the robot's distance to other objects [Chong *et al.*, 2015]. By computing the time taken for the pulse to reflect and return whilst having the laser rotate perpendicular to the robot's reference frame, it is possible to obtain sensor readings in a full 360-degree circle.

Alternative sensors that can be used for SLAM include cameras and RGB-D sensors. The subset of algorithms that use these sensors is known as *V-SLAM*, which is an upcoming research area. Being a problem where solutions can greatly benefit from *sensor fusion* (the combination of sensory data to reduce uncertainty) many SLAM algorithms exploit a multitude of sensors available on the system. The most common examples include *odometry* sensors (estimating movement based on the rotation of the robot's wheels), gyroscopes, accelerometers, and magnetometers. These last three comprise the robot's *inertial measurement unit* (IMU).

3 Related Work

3.1 Inspirational Empirical Research in Robotics Software

Malavolta *et al.* [2021] identified and performed an empirical evaluation of *green architectural tactics* applied in robotics software. Evaluation involves conducting measurements of a robot's energy consumption in an experimental environment. This has been an inspiration for the experiment design and execution of the present work, though the subjects and contextual implications of the evaluation are different. Whereas Malavolta *et al.* [2021] considers general software engineering patterns that can be applied to improve energy, the present work is devoted to SLAM algorithms and their ancillary domain-specific design issues.

3.2 SLAM Quality Evaluation

There are several works in the literature that focus on evaluating quality attributes of SLAM algorithms, such as performance and accuracy. Numerous of those studies likewise

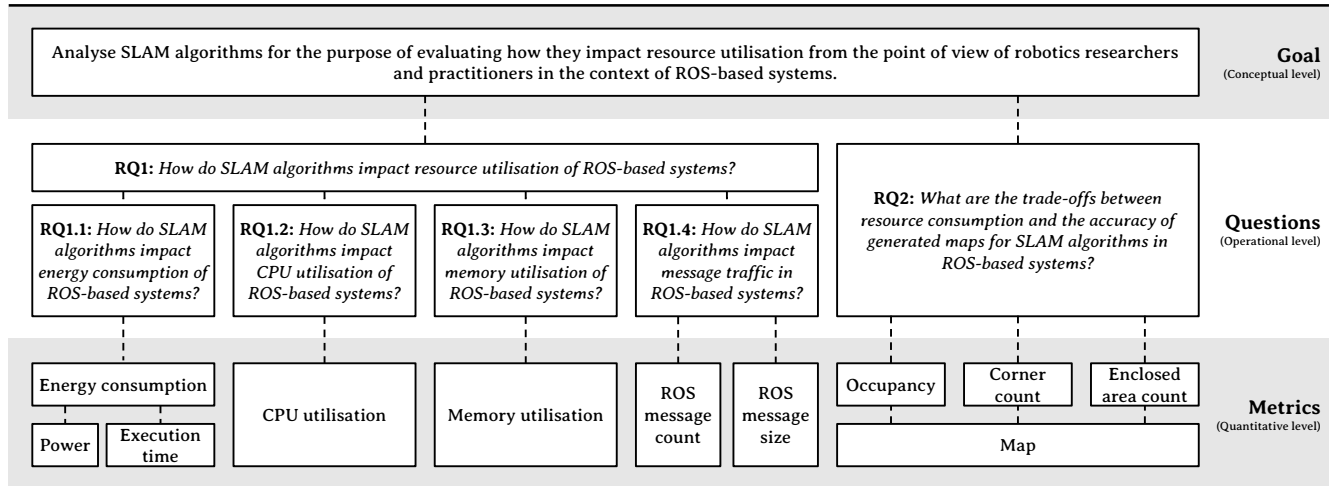


Figure 1. Visualization of the GQM tree Basili *et al.* [1994] used for experiment definitions.

ours investigate laser-based algorithms [Sankalprajan *et al.*, 2020; Abdelrasoul *et al.*, 2016; Santos *et al.*, 2013; Rojas-Fernández *et al.*, 2018; Le *et al.*, 2018; Roesler and Ravindranath, 2020; Andert and Mosebach, 2019; Duchoň *et al.*, 2019; Aerts and Demeester, 2017]. Contrary to our investigation, most of those works only analyze performance measurements. An exception is the work by Santos *et al.* [2013], which additionally considers CPU load as a metric; however, their experimental environment and navigation goals are different from ours and the extent of their experimentation is narrower, without considering SLAM algorithms' reconfiguration.

Various other studies have proposed methods to evaluate the performance of SLAM algorithms. The most prevalent of these is *SLAMBench* [Nardi *et al.*, 2015]: a software framework that facilitates quantitative experimental research investigating trade-offs in several aspects of SLAM systems. These include performance, accuracy, and energy consumption. The framework is expanded in other studies [Zia *et al.*, 2016]. Another study that facilitates research evaluating SLAM algorithms is performed by Sturm *et al.* [2012]. By recording a large set of input data and recording accurate *ground truth* trajectories, the authors aim to create a new benchmark that is suitable for repeatable experiments and objective comparisons. While the above benchmarks have proven to be useful in a multitude of studies and their goals align well with those of the present work, both of them are only compatible with RGB-D SLAM algorithms. Conversely, the present work considers laser-based methods. It thus complements these existing SLAM benchmarks.

While the above studies do investigate the *quality* of the maps created by SLAM, they often do not consider the associated resource utilization of the algorithms. Some exceptions present measurements of CPU utilization for several algorithms [Santos *et al.*, 2013; Abdelrasoul *et al.*, 2016; Rojas-Fernández *et al.*, 2018], and others with an additional memory utilization metric [Le *et al.*, 2018]. Energy usage, an important metric in the present work, is not evaluated in any of the studies.

Two recent works are also closely related to ours, as they also compare different SLAM algorithms. Ngoc *et al.* [2023] conducts a comparative analysis of three SLAM algorithms,

i.e., GMapping, Cartographer SLAM and SLAM_toolbox, within the ROS2 framework, evaluating both map accuracy and robot navigation efficiency. Nguyen *et al.* [2022] focuses on deploying SLAM techniques in indoor environments, particularly to aid first responders in environments where odometry is unavailable due to manual scanning operations. They compare ROS-based SLAM algorithms, including GMapping, HectorSLAM, and Cartographer. While these works concentrate exclusively on the performance of SLAM algorithms in specific environments, our study emphasizes the trade-offs involved. In our investigation, performance is just one of the dependent variables, assessed alongside resource consumption. Moreover, each of these studies evaluates SLAM algorithms in different environmental contexts.

If we look only at Brazilian community venues, we also observe significant contributions to evaluating SLAM algorithms. Costa and Colombini [2023] reported a reduction in estimation error for Visual SLAM (V-SLAM) ranging from 2.81% to 15.98% by employing semantic filtering to identify dynamic objects. De Mello Gai *et al.* [2023] demonstrated that inconsistencies in multi-robot localization are often due to the propagation of sensor mis-measurements and noise. To address this issue, they proposed combining data from various sensors and algorithms, such as odometry, Inertial Measurement Units (IMU), and Adaptive Monte Carlo Localization (AMCL). Meanwhile, Alcalde *et al.* [2022] introduced the use of Deep Reinforcement Learning (DRL) to enhance SLAM, which significantly improves robot navigation in complex environments. Finally, the work of Lins *et al.* [2020] is one of the most related to our research, which explored SLAM with different parameterizations; however, their focus was on robot description parameters rather than the ROS system. None of those evaluations has a similar goal to ours, and the nature of their studies is also distinct since they exclude a comparison among SLAM algorithms and the impact of their parametrization.

3.3 SLAM Resource Usage Evaluation

There are a few studies that aim to investigate the resource usage of SLAM algorithms. Vaussard *et al.* [2012] performed

Table 1. Experiment *goal definition* based on the GQM framework [Basili *et al.*, 1994].

Analyse	SLAM algorithms
For the purpose of	Evaluation
With respect to	Resource utilisation
From the point of view of	Robotics researchers and practitioners
In the context of	ROS-based systems
Result Analyse SLAM algorithms to evaluate how they impact resource utilization from the point of view of robotics researchers and practitioners in the context of ROS-based systems.	

an empirical evaluation of energy consumption in commercially available autonomous vacuum cleaners. Some of the studied subjects employed SLAM algorithms for mapping the room and following a predetermined path while cleaning, whereas other models adopted a more naive approach of random traversal. While the SLAM robots are computationally more complex, their increased efficiency resulted in a shorter duration of traversing the room and ultimately lower energy consumption. While both this study and the present work consider energy aspects of robotic systems using SLAM, Vaussard *et al.* [2012] evaluated robotic systems as a whole (including possible hardware differences that are not accounted for). The present work isolates the effects of SLAM algorithms in *software* by using the same robotic system throughout the entire experiment. Furthermore, the commercially available systems act as a 'black box' without disclosing the characteristics of their specific SLAM implementation, whereas the present work uses publicly available packages as subjects.

Peng *et al.* [2019] evaluated the energy consumption of various SLAM algorithms in the context of GPU-accelerated robotic systems. To collect power measurements of the subjects under study, the same INA219 sensor as in the present work is used. In contrast to the present work, the authors put a special emphasis on the impact of hardware on *visual* SLAM (V-SLAM) algorithms.

Zhao *et al.* [2024] compare the performance of two SLAM methods, i.e., ORB-SLAM3 and SC-LeGO-LOAM, using synthetic datasets that integrate data from 3D LiDAR and RGB-D cameras. They provide a detailed comparison of localization accuracy, mapping stability, and computational resource usage, revealing that Visual SLAM is more resource-intensive but performs better in indoor environments, while LiDAR SLAM is better suited for outdoor applications. Different from our study, they focus on the type of sensor data instead of on the SLAM algorithms.

Dordević *et al.* [2023] investigate computation offloading over WiFi (to a computer at the edge), focusing on the trade-offs between performance and energy consumption. Through empirical analysis, the authors explore how offloading affects system efficiency, providing insights into balancing computational demands with energy usage in robotic operations. Despite investigating the trade-offs between performance and resource usage, the authors do not compare different SLAM algorithms.

4 Experiment Definition

Figure 1 illustrates the hierarchical relationships between the goal, questions, and metrics in this experiment, following the *Goal Question Metric* (GQM) framework proposed by Basili *et al.* [1994], which has been largely used in recent empirical software engineering studies [Alharbi and Alshayeb, 2024; Castaño *et al.*, 2023; Koziolok *et al.*, 2020], including studies on software energy efficiency [Lee *et al.*, 2024] and performance [Cabane and Farias, 2024]. GQM is also referenced as a viable framework for accessing software development processes [Wohlin *et al.*, 2012].

Table 1 formally defines the goal of this experiment according to the GQM framework, which leads to the following research questions:

- **RQ1:** *How do SLAM algorithms impact resource utilization of ROS-based systems?*

SLAM algorithms tend to be computationally intensive leading to high CPU and memory usage, which depend on the implemented technique, e.g., graph-based or particle-filter-based SLAM. Therefore, the hypothesis guiding the above-mentioned research questions is that using different SLAM algorithms may lead to significantly distinct resource usage.

- **RQ2:** *What are the trade-offs between resource consumption and the accuracy of generated maps for SLAM algorithms in ROS-based systems?*

The main hypothesis that leads to **RQ2** is that algorithms that consume more resources should lead to more accurate maps. However, as discussed in **RQ1**, resource usage may be directly impacted by the used technique, which can be inefficient and resource-demanding or the opposite. In this case, revealing SLAM algorithms that achieve high accuracy with low resource consumption would work as an important baseline for future design decisions.

RQ1 is explored through the following sub-questions:

- **RQ1.1:** *How do SLAM algorithms impact energy consumption of ROS-based systems?* - Energy consumption is a critical factor for mobile robots where battery life is often limited [Albonico *et al.*, 2021].
- **RQ1.2:** *How do SLAM algorithms impact CPU utilization of ROS-based systems?* - High CPU utilization can lead to system bottlenecks, slower processing times, and increased energy consumption, especially for robots, which usually have restricted computation power [Partap *et al.*, 2022].
- **RQ1.3:** *How do SLAM algorithms impact memory utilization of ROS-based systems?* - Memory utilization is

another resource constraint in mobile robotic systems with limited computation power, and it may be restrictive for SLAM algorithms, which store and process large amounts of data [Suger *et al.*, 2014].

- **RQ1.4:** *How do SLAM algorithms impact message traffic in ROS-based systems?* - Given the significant amount of data generated by SLAM algorithms, which then must be communicated between different ROS nodes, may have a direct impact on the ROS system's latency, bandwidth utilization, and overall performance.

Answers to these questions can aid researchers working on SLAM algorithms in the context of ROS-based systems in making resource-aware design choices. Furthermore, these answers can aid practitioners in deciding which SLAM algorithm to choose when architecting ROS-based systems where resources (such as energy) are limited.

4.1 Metrics

The research questions are answered after metrics and measurement units listed in Table 2, which come from the three main measurements described in the sequence. The devices and software artifacts used in the measurements are described in Section 6.2.

4.1.1 Resource Utilization

Energy consumption, *CPU utilization*, *memory utilization* are directly linked to answering **RQ1** as they provide a comprehensive view of the resource utilization of ROS-based systems when running SLAM algorithms, and are then used to compare with map accuracy when answering **RQ2**.

While *CPU* and *memory utilization* have been largely used in empirical software engineering work in the literature, *energy consumption* deserves a further explanation. That metric is computed as $E = \frac{P \times T}{10^6}$ where P denotes the average power consumption throughout a mission, T denotes the duration of the mission, and the division by 10^6 signifies a conversion from micro-joules to Joules.

4.1.2 ROS Message Traffic

This metric also relates to **RQ1** since message traffic impacts the system's communication overhead, which is an important observation when investigating the causes of resource utilization behaviors.

By recording all traffic between the ROS nodes in the evaluated system during the experiment, the *message count* and *total size* can be derived. These metrics can provide insight into how the implementations of the algorithms utilize the ROS communication middleware to transmit (intermediate) data among nodes.

4.1.3 Map Accuracy

This is the key metric for **RQ2**, used to evaluate the quality of the SLAM output, where generating maps is their main goal. Accurate mapping ensures that the system can adapt and operate in different environments, while inaccuracies can lead

to navigation and localization errors, resulting in mission failures.

To assess the accuracy of a SLAM algorithm, a method that comes to mind is an immediate comparison between the produced map and a reference taken from the real environment (which is known to be fully correct). This method is explored by Yagfarov *et al.* [2018] by using computer vision techniques to align the obtained map with a *ground truth*. A downside of this method is that a map's rotational orientation can affect the extent to which alignment is possible, even though this does not necessarily mean that the map's accuracy is lacking. Furthermore, it is not always possible to obtain a ground truth of the environment in which an experiment takes place.

As an alternative, this study adopts a set of accuracy metrics proposed by [Filatov *et al.*, 2017], focused on evaluating 2D SLAM map quality based on three metrics computed from the obtained maps independent of any underlying reference maps. These metrics are the *occupancy ratio*, *corner count* (obtained from Harris' corner detection algorithm [Harris *et al.*, 1988]) and *enclosed area count* (obtained from Suzuki's algorithm [Suzuki *et al.*, 1985]). These metrics can be used to compare maps with one another under the assumption that the maps themselves contain comparable coverage of the area (i.e., they both represent roughly the same environment). Later relevant studies also adopted these metrics and proved their suitability in a context similar to this study [Su *et al.*, 2020; Dhaoui and Rahmouni, 2022; Samarakoon *et al.*, 2022].

5 Experiment Planning

5.1 Subject Selection

As introduced in Section 2.2, there exists a multitude of packages in ROS that address the problem of SLAM. Since the laser range finder is among the most common sensors for this task [Chong *et al.*, 2015], four laser SLAM implementations available in ROS are selected as subjects for this study, as shown in Table 3: (i) Cartographer, (ii) Gmapping, (iii) Hector SLAM and (iiii) Karto. One additional ROS package offering laser SLAM functionality exists by the name of *SLAM Toolbox*, yet this is an extension of Karto and has been omitted from this study in favor of the original. An overview comparing some characteristics of each algorithm is provided in Table 3. All algorithms are *passive*, meaning that they do not influence the movement of the robot while traversing an environment [Stachniss *et al.*, 2016]. Furthermore, all algorithms use an *occupancy grid* to represent the map, which is a discrete data structure where each cell is assigned a probability of being occupied.

Cartographer is the youngest of all evaluated algorithms, introduced in 2016 by Google researchers Hess *et al.* [2016]. Google itself uses Cartographer for mapping building interiors in *Street View*. This algorithm falls within the graph SLAM paradigm. The algorithm iteratively matches incoming laser scans with a recent *submap*, thus scan matching is dependent on recent scans only. A major difference compared to the other algorithms is that Cartographer also uses

Table 2. Experiment metrics.

Metric	Unit	Frequency	Description
Power	Milliwatts (mW)	200 Hz	The average electrical power consumed by the robotic system during a mission.
Execution Time	Milliseconds (ms)	-	The time taken to execute a mission.
Energy Consumption	Joules (J)	-	The total amount of energy consumed during a mission.
CPU Utilization	Percentage (%)	50 Hz	The ratio of CPU capacity utilized by the robotic system during a mission.
Memory Utilization	Megabytes (MB)	50 Hz	The amount of memory utilized by the robotic system during a mission.
ROS Message count	-	Continuous	The number of messages exchanged between ROS nodes on the robotic system during a mission.
ROS Message Total Size	Bytes (B)	Continuous	The total size of messages exchanged between ROS nodes on the robotic system during a mission.
Map Occupancy	Percentage (%)	-	The ratio of occupied cells in the final map [Filatov <i>et al.</i> , 2017].
Map Corner Count	-	-	The number of pixels in the final map that represent corners [Filatov <i>et al.</i> , 2017].
Map Enclosed Area Count	-	-	The number of enclosed areas in the final map [Filatov <i>et al.</i> , 2017].

IMU data to perform sensor fusion [Aerts and Demeester, 2017].

Gmapping (short for *grid-mapping*) is a particle filter SLAM algorithm proposed by Grisetti *et al.* [2007]. By exploiting the same Rao-Blackwellized particle filters and incorporating adaptive resampling, the algorithm improves upon the prior PF-SLAM FASTSLAM [Montemerlo *et al.*, 2002], extending it to grid maps. Odometry data is combined with recent observations to compute a more accurate sampling distribution, which reduces the uncertainty. This means that a lower number of particles is required to obtain good results, which contributes to computational efficiency.

Hector SLAM is a SLAM algorithm proposed by Kohlbrecher *et al.* [2011] which also belongs to the particle filter paradigm. By considering limited resource utilization as an explicit requirement, the authors aimed to achieve accurate mapping capabilities even on low computational resources. A primary design choice made to enable this is Hector's strategy of directly mapping observations from the laser scans onto the map in the best-aligned location. Subsequent map optimization is not required, thereby saving computational effort. Furthermore, Hector does not apply any *sensor fusion* techniques, whereas the other algorithms incorporate odometry data in map creation and pose estimation. This is a beneficial property in case of unreliable odometry sensors, or an environment that does not allow for accurate odometry sensing. Furthermore, Hector lacks an explicit *loop closure* feature, which the

other algorithms employ to re-calibrate the estimated robot pose when encountering landmarks that have been visited before.

Karto SLAM as proposed by Konolige *et al.* [2010] is another graph-based SLAM approach. It maintains a pose graph to which sparse pose adjustment is applied for both scan matching and explicit loop closure. As opposed to some of the other algorithms, Karto uses the complete path and map when computing a new map instead of only using the most recent data.

While they provide a wide range of parameters that can be tuned (e.g., the amount of particles used in a filter, or proximity thresholds used for loop closure) this experiment uses the default configuration of each algorithm.

5.2 Experimental Variables

The experimental variables for this experiment are listed in Table 4. The remainder of this section describes the motivation for choosing these variables and their respective values.

5.2.1 Independent variables

The most important *independent* variable under investigation in this study is the used *SLAM algorithm*. Each of them offers a broad range of configuration parameters, which may affect their performance and resource consumption. To this extent three parameters that all four implementations share

Table 3. Experiment subjects and their properties.

Name	Type	First released	Uses odometry	Uses IMU	Default linear update threshold	Default angular update threshold	Default map resolution
Cartographer [Hess <i>et al.</i> , 2016]	Graph	2016	Yes	Yes	0.2 m	0.0174 rad	0.05 m/pixel
Gmapping [Grisetti <i>et al.</i> , 2007]	Particle filter	2007	Yes	No	1.0 m	0.5 rad	0.05 m/pixel
Hector [Kohlbrecher <i>et al.</i> , 2011]	EKF/PF	2011	No	No	0.4 m	0.9 rad	0.05 m/pixel
Karto [Konolige <i>et al.</i> , 2010]	Graph	2010	Yes	No	0.2 m	0.174 rad	0.05 m/pixel

are selected as additional experimental variables: *linear update threshold*, *angular update threshold*, and *map resolution*. An additional configuration variable is the *map resolution*, which refers to the real-world distance one pixel in the obtained map represents. These variables were chosen for their relevance in influencing SLAM performance and are critical for assessing both, accuracy and computational efficiency. Each of these variables is explained in detail in the sequence.

SLAM algorithm: The possible values for this variable correspond to the four representative algorithms that have been selected as the subjects of this study, as motivated in Section 5.1. Using multiple SLAM algorithms or configurations could provide insight into which approach is more robust across two preset navigation scenarios, enhancing generalization.

Linear and angular update thresholds: The first two configuration variables are the *linear update threshold* and the *angular update threshold*. New laser scans are only processed by the SLAM algorithm once the robot has either traveled a distance or performed a rotation that exceeds these respective values. The default configuration for each of the algorithms under study is listed in Table 3. As a method to objectively select two possible values for both variables to use during the experiment, we considered the minimum and maximum values among all algorithms. Picking these extreme values ensures that (i) the default value of all algorithms lies between the possible values and (ii) the potential observable differences among how these variables impact the studied metrics are the largest.

The *linear update threshold* defines how often the map is updated based on the robot's movement. A small threshold may result in frequent updates and higher computational load, while a larger threshold may miss map details. Varying this parameter value may affect mapping accuracy in different environments, with higher thresholds potentially reducing map precision in complex areas. However, this is not considered in this stage of experiments.

The *angular update threshold* affects map update frequency based on the robot's rotation. Similarly to *linear update threshold*, on one hand, small values may lead to more accurate maps but higher computational costs. On the other hand, a too-high angular threshold could miss critical rotational information in environments with frequent turns, affecting localization accuracy and overall map quality.

Map resolution: all algorithms share the same default value for this variable (0.05 m/pixel), hence the aforementioned selection technique does not hold here. Considering the default is relatively high, the second possible value for this variable in the experiment is set to half the default resolution. This is motivated by the potential of tweaking the resolution as a means of reducing resource consumption, should an application context allow for coarser maps.

The choice of map resolution determines the granularity of the environment representation. Lower resolutions may make maps less accurate in complex environments, while higher resolutions may overload computational resources, affecting real-time performance.

Arena: since the robot's environment may have an influence on the dependent variables under study, the *arena* in which a mission is executed is considered as the final independent

variable. The experiment is performed in two different arenas: one where the robot can traverse a circular path throughout the arena (starting and ending at the same location) and another which requires a point-to-point trajectory. The design of these arenas is further motivated in Section 6.3.

SLAM system may perform differently in more dynamic (e.g., with obstacles) or less structured environments. Including a wider range of environments could have provided more comprehensive results. However, we choose to keep the experiment simple, going from small scenarios with no obstacles, resulting in fewer variables to be analyzed, which potentially improves understandability.

5.2.2 Static variables

The *static* variables, as listed in Table 4, are the same for all runs. They are fixed in order not to increase the experiment's complexity further.

At the core of the experiment, the *robot platform* used is the *TurtleBot3 Burger* by ROBOTIS². This ground robot is widely used in research, due to its compact form factor and extensive sensor capabilities [Dordevic *et al.*, 2021]. This includes a 360-degree laser distance sensor (LIDAR), which is used by the SLAM algorithms to scan the environment.

The Single Board Computer (SBC) mounted on the robot is a *Raspberry Pi 4* with 8 Gigabytes of RAM. Given its computing power limitations, the robot's linear velocity is set to a consistent 0.10 m/s throughout the entire mission.

The ROS distribution used for this experiment is *ROS 1 Noetic Ninjemys*, running on Ubuntu 20.04. This is the most recent long-term support ROS 1 distribution at the time of writing. While the robot platform is also capable of running ROS 2 packages, the limited availability of the algorithms under investigation in this study resulted in an exclusive focus on ROS 1.

Due to the portable nature of packages in the ROS ecosystem, it is possible to execute the SLAM algorithms either onboard the robot or offload them to a remote PC. Since this study focuses on the immediate impact of said algorithms on the robot's resource utilization, only the configuration where the SLAM algorithm is executed on the robot itself is considered. The same holds for the *ROS master node*.

5.2.3 Dependent variables

All the metrics, as introduced and motivated in Section 4.1, form the *dependent* variables of this study.

5.3 Experimental Hypotheses

Fifteen formal hypotheses are compiled for all the dependent variables in this study. Their assertion should help us answer the research questions.

$$H1 \quad H_0^1 : \tau_1 = \tau_2 = \tau_3 = \tau_4 \quad H_a^1 : \exists i, j \text{ such that } \tau_i \neq \tau_j$$

Where $\tau_n \dots \tau_4$ denote the effect of Cartographer, Gmapping, Hector, and Karto, respectively;

²<https://emanual.robotis.com/docs/en/platform/turtlebot3/features/>

Table 4. Experiment variables.

Type	Name	Category	Description
Independent variables (Input)	Algorithm	Nominal	One of the algorithms which are subjects of this study. <i>Cartographer, Gmapping, HectorSLAM, Karto</i>
	Map resolution	Ordinal	The resolution of the produced map. <i>0.05 m/pixel (high), 0.10 m/pixel (low)</i>
	Linear update threshold	Ordinal	The minimum linear distance traveled by the robot to trigger processing of new laser scans. <i>1.0 m (high), 0.2 m (low)</i>
	Angular update threshold	Ordinal	The minimum angular pose difference of the robot to trigger processing of new laser scans. <i>0.9 rad (high), 0.0174 rad (low)</i>
	(Blocking factor) Arena	Nominal	The environment to be mapped, defining the trajectory of the robot. <i>Circular, Point to point</i>
Static variables (Same for all runs)	Robot platform	Nominal	The robot that runs the algorithm <i>ROBOTIS TurtleBot3 Burger</i>
	Robot SBC	Nominal	The single board computer mounted on the robot. <i>Raspberry Pi 4 model B</i>
	Linear velocity	Ordinal	The speed at which the robot travels in the linear direction. <i>0.10 m/s</i>
	ROS distribution	Nominal	The ROS distribution installed on the robot's SBC and remote PC. <i>ROS 1 Noetic Ninjemys (Ubuntu 20.04)</i>
	SLAM and ROS master host	Nominal	The host machine where the SLAM and ROS master nodes are run. <i>Onboard the robot</i>
Dependent variables (Output)	Power	Ordinal	The power consumed by the robotic system. <i>in Milliwatts (mW)</i>
	Execution time	Ordinal	The time taken to execute a mission. <i>in seconds (s)</i>
	Energy consumption	Ordinal	The total amount of energy consumed. <i>in Joules (J)</i>
	CPU utilization	Ratio	The ratio of CPU capacity utilized by the robotic system. <i>in percent (%)</i>
	Memory utilisation	Ordinal	The amount of memory utilised by the robotic system. <i>in Megabytes (MB)</i>
	ROS message count	Ordinal	The number of messages exchanged between ROS nodes on the robotic system.
	ROS message size	Ordinal	The size of messages exchanged between ROS nodes on the robotic system. <i>in bytes (B)</i>
	Map occupancy	Ratio	The ratio of occupied cells in the final map [Filatov <i>et al.</i> , 2017]. <i>in percent (%)</i>
	Map corner count	Ordinal	The number of pixels in the final map that represent corners [Filatov <i>et al.</i> , 2017].
	Map enclosed area count	Ordinal	The number of enclosed areas in the final map [Filatov <i>et al.</i> , 2017].

- H2** $H_0^2 : v_1 = v_2$ $H_a^2 : v_1 \neq v_2$
Where v_1 and v_2 denote the effect of a high and low linear update threshold, respectively;
- H3** $H_0^3 : \phi_1 = \phi_2$ $H_a^3 : \phi_1 \neq \phi_2$
Where ϕ_1 and ϕ_2 denote the effect of a high and low angular update threshold, respectively;
- H4** $H_0^4 : \chi = \chi_2$ $H_a^4 : \chi_1 \neq \chi_2$
Where χ_1 and χ_2 denote the effect of a high and low map resolution, respectively.

The following hypotheses are formulated for the interaction effects of the factors, denoted using the same symbols as for the main effects between parentheses:

- H5** $H_0^5 : (\tau v)_{i,j} = 0 \forall i, j$ $H_a^5 : \exists i, j \text{ such that } (\tau v)_{i,j} \neq 0$
- H6** $H_0^6 : (\tau \phi)_{i,j} = 0 \forall i, j$ $H_a^6 : \exists i, j \text{ such that } (\tau \phi)_{i,j} \neq 0$
- H7** $H_0^7 : (\tau \chi)_{i,j} = 0 \forall i, j$ $H_a^7 : \exists i, j \text{ such that } (\tau \chi)_{i,j} \neq 0$
- H8** $H_0^8 : (\phi v)_{i,j} = 0 \forall i, j$ $H_a^8 : \exists i, j \text{ such that } (\phi v)_{i,j} \neq 0$
- H9** $H_0^9 : (\phi \chi)_{i,j} = 0 \forall i, j$ $H_a^9 : \exists i, j \text{ such that } (\phi \chi)_{i,j} \neq 0$
- H10** $H_0^{10} : (v \chi)_{i,j} = 0 \forall i, j$ $H_a^{10} : \exists i, j \text{ such that } (v \chi)_{i,j} \neq 0$
- H11** $H_0^{11} : (\tau v \phi)_{i,j,k} = 0 \forall i, j, k$ $H_a^{11} : \exists i, j, k \text{ such that } (\tau v \phi)_{i,j,k} \neq 0$
- H12** $H_0^{12} : (\tau v \chi)_{i,j,k} = 0 \forall i, j, k$ $H_a^{12} : \exists i, j, k \text{ such that } (\tau v \chi)_{i,j,k} \neq 0$
- H13** $H_0^{13} : (\tau \phi \chi)_{i,j,k} = 0 \forall i, j, k$ $H_a^{13} : \exists i, j, k \text{ such that } (\tau \phi \chi)_{i,j,k} \neq 0$
- H14** $H_0^{14} : (v \phi \chi)_{i,j,k} = 0 \forall i, j, k$ $H_a^{14} : \exists i, j, k \text{ such that } (v \phi \chi)_{i,j,k} \neq 0$
- H15** $H_0^{15} : (\tau v \phi \chi)_{i,j,k,l} = 0 \forall i, j, k, l$ $H_a^{15} : \exists i, j, k, l \text{ such that } (\tau v \phi \chi)_{i,j,k,l} \neq 0$

5.4 Experiment Design

Resulting of the variables and hypotheses stated above, the experiment follows a multiple-factor, multiple-treatment design (MFMT). Its five factors correspond to the independent variables listed in Table 4. The *algorithm* factor has 4 treatments: Cartographer, Gmapping, Hector, and Karto. Conversely, the configuration factors *linear update threshold*, *angular update threshold* and *map resolution* each have two treatments: low and high.

The *arena* is considered as a blocking factor. The reason for this is twofold: (i) it is not part neither of the algorithm nor any configuration onboard the robotic system, and (ii) different arenas require a different amount of time to traverse whilst maintaining an identical velocity, which has an inherent impact on the energy consumed (making a direct comparison unbalanced). Two treatments are selected for the arena (i.e., circular and point-to-point), which are further elaborated on in Section 6.3.

To facilitate the investigation of the interactions between all factors, the experiment is set up in a full-factorial design, meaning that **all possible combinations of treatments are evaluated**. Considering that resource consumption aboard

real-world systems is particularly volatile, this design is fortified by two precautions: (i) each combination of treatments is repeated 5 times, and (ii) the order in which all combinations of treatments are evaluated is fully randomized. In conclusion, the experiment adds up to 320 runs:

$$4 \times 2^4 \times 5 = 320 \quad (2)$$

5.5 Data Analysis

Upon completion of the experiment, the collected measurements are analyzed to answer the research questions. For each combination of the metrics under study and the arena, the analysis phase can be divided into three sequential stages: (i) *data exploration*, (ii) *statistical tests*, and (iii) *post-hoc tests*, which are explained in the sequence.

5.5.1 Data exploration

The goal of this first stage is to gain insights about the data. Descriptive statistics, histograms, and box plots show the distribution of values and highlight potential differences among groups in the evaluation. By plotting the obtained measurements as a time series their variation throughout the mission can be observed. The results obtained from this stage aid in the explanation of observations made during subsequent stages.

5.5.2 Statistical tests

The hypothesis tests, as stated in Section 5.3, rely on an *analysis of variance* (ANOVA), where the *factorial* ANOVA is applied, i.e., two or more independent factors and a single response variable. Since this method relies on a linear model, the following assumptions shall be met for the results of the analysis to be valid [Snedecor *et al.*, 1968]:

1. **Independence of observations** – An observation provides no information about any another observation;
2. **Normality** – The model's *residuals* are normally distributed;
3. **Homogeneity of variances** – The variance among each group in the analysis is identical.

By the design of the experiment, the first assumption is always met. To verify whether assumption (2) holds, a linear model is fitted to the data, and residuals are computed. Normality is then checked using visual inspection of the Q-Q plot of residuals in combination with a Shapiro-Wilk test [Shapiro and Wilk, 1965]. The outcome of these tests should reject the normal distribution of residuals, then the model is fitted again after applying various transformations to the obtained measurement data and the checks are applied as described before. Homogeneity of variances (3) is verified by visual inspection of the variance plotted against the mean for each group under study and complemented with Levene's test [Brown and Forsythe, 1974].

If the above checks reveal that the assumptions for the parametric ANOVA are not satisfied, then the analysis resorts to a non-parametric alternative known as PERMANOVA [Anderson, 2001]. This method utilizes an analogous test statistic to Fischer's F-ratio (as in the original

ANOVA) and relies on permutation tests to compute p-values. The fact that this method lacks formal assumptions is a major advantage to its usability in the context of this study. All the tests are evaluated using a confidence interval of 95% ($\alpha = 0.05$), as is common in empirical software engineering research.

5.5.3 Post-hoc tests

Significant main effects or higher-order interactions that emerge from the statistical tests elicit an opportunity for additional investigation using *post-hoc* tests. The rationale for choosing a posthoc test is as follows:

- If a significant three or four-way interaction exists, a follow-up ANOVA is carried out for both levels of the last factor;
- If a significant two-way interaction exists where neither factors are involved in higher order interactions, *Tukey's honestly significant difference (HSD) test* [Abdi and Williams, 2010] is performed. The assumptions for this test are identical to those for the ANOVA with the addition of the normality of the data per group. Dunn's test [Dunn, 1964] is selected as a non-parametric alternative;
- If there exists a significant simple main effect for the algorithm factor Tukey's HSD test or Dunn's test are used to investigate the differences among the 4 treatments (which does not follow from the (PERM)ANOVA.

Where applicable, post-hoc tests are corrected for multiple comparisons using the *Benjamini-Hochberg* correction [Benjamini and Hochberg, 1995].

5.6 Experiment Replicability

A full *replication package* of this study is publicly available on GitHub³, enabling the reader to verify and reproduce all experimental observations and conclusions presented in this paper, which includes: **all the raw data** with measurements, recorded ROS traffic, and log files obtained throughout the experiment; the **python and R scripts** used to analyze the data and produce the figures in this paper; the **experiment configuration files** used to (partially) automate the execution of the experiment; and the **ROS packages** used for profiling resource utilization via services.

6 Experiment Execution

6.1 Experiment Infrastructure

An overview of the experiment's infrastructure in terms of its components and their interactions is provided in Figure 2.

6.2 Instrumentation

This section describes all hardware and software used in the experiment and motivates any choices that were made in the process of selecting these components.

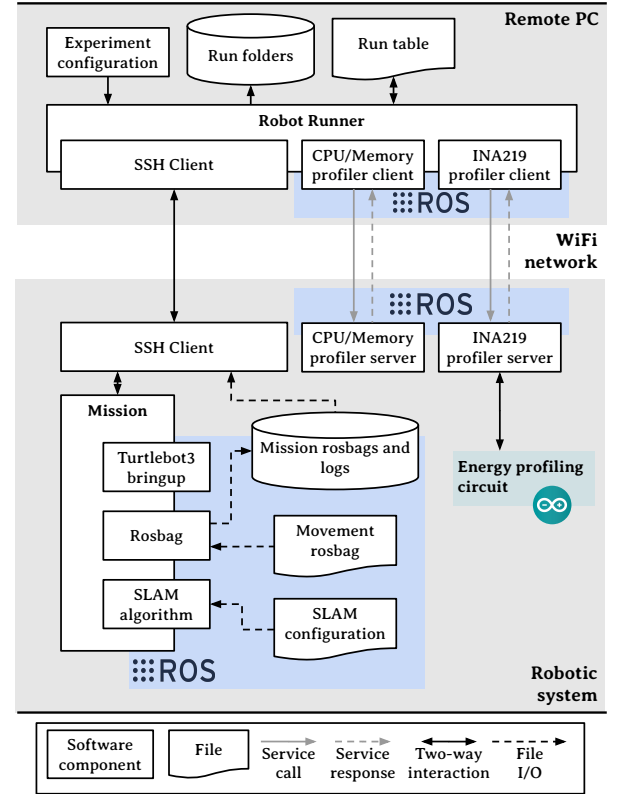


Figure 2. Deployment of experimental software artifacts divided into two groups, i.e., ROS robotic system and Robot Runner (RR) orchestrator, running on distinct devices connected over the WiFi network. The robotic system is remotely stimulated via SSH by an RR's plugin, while resource usage profilers monitor the robotic system via dedicated ROS packages.

6.2.1 Hardware

The experiments are orchestrated by sending commands to the robotic system over the local and dedicated WiFi network. All the experiments run on a laptop with an Intel Core i5-6200U processor and 8 Gigabytes of memory. The following hardware is used for setting up the robotic system, as depicted in Figure 2:

Table 5. TurtleBot3 Burger system specification.

Variable	Value
Microcontroller	32-bit ARM Cortex-M7 with FPU (216 MHz, 462 DMIPS)
Battery	LiPo 11.1V 1800mAh 5C
Laser distance sensor	360 Laser Distance Sensor LDS-01
Size (L x W x H)	138mm x 178mm x 192mm
Maximum linear velocity	0.22 m/s
Maximum angular velocity	2.84 rad/s

- **ROBOTIS TurtleBot3 Burger (robotic system)⁴** – The robotic system used in the experiment. It is chosen due to its widespread usage in existing research, compatibility with ROS, and extensive sensor capabilities [Dordevic *et al.*, 2021]. The SLAM algorithms use

³Hidden for double-blind review.

⁴<https://emanual.robotis.com/docs/en/platform/turtlebot3/features/>

the 360-degree laser distance sensor to scan the environment. A more detailed system specification of the TurtleBot is listed in Table 5.

- **Raspberry Pi 4 model B (robot SBC)** – The single board computer that runs ROS and controls the robot. While the de-facto SBC of the TurtleBot is a Raspberry Pi 3 model B, its successor is used in this experiment. This unit with 8 gigabytes of RAM ensures sufficient computational power and memory for running the SLAM algorithms under every evaluated configuration.
- **NetGear R6220 WiFi Router** – Facilitates an isolated WiFi network to which the robot and remote PC are connected during the experiment.
- **Custom energy profiling circuit** – It is a hardware addition to the TurtleBot that enables measurement of the robot’s power consumption, which was originally proposed by Swanborn [2020]. For more details, including its coding, there is a public replication packaged⁵.

The energy profiling circuit is shown in Figure 3, consisting of the following main components:

- **Arduino Nano⁶** – Low-energy computational unit responsible for collecting the measurements from the power sensor. A serial connection to the robot’s SBC is used for receiving commands and returning measurements after profiling is stopped.
- **AdaFruit INA219 power sensor⁷** – This power measurement sensor is specifically designed for use with microcontrollers and has a measuring precision of 1%. This ensures both a low energy consumption profile of the sensor itself, as well as sufficient accuracy to substantiate the validity of the findings of this study. The sensor has proven to be suitable for similar experimental setups in existing studies [Malavolta *et al.*, 2021; Peng *et al.*, 2019].
- **TinyTronics SD Card module** – Simple module for SD card I/O on the Arduino Nano. Used to write measurements to an 8GB SD card while profiling.

6.2.2 Software

The software components that are used for this experiment are:

- **ROS Noetic Ninjenkis⁸** – The ROS distribution used for this experiment, running on Ubuntu 20.04.
- **ROS SLAM packages** – Implementation packages of the algorithms that are the subjects of this study: *cartographer*⁹, *gmapping*¹⁰, *hector_mapping*¹¹ and *slam_karto*¹².

⁵Hidden for double-blind review.

⁶<https://www.arduino.cc/en/pmwiki.php?n=Main/ArduinoBoardNano>

⁷<https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ina219-current-sensor-breakout.pdf>

⁸<http://wiki.ros.org/noetic>

⁹<http://wiki.ros.org/cartographer>

¹⁰<http://wiki.ros.org/gmapping>

¹¹http://wiki.ros.org/hector_mapping

¹²http://wiki.ros.org/karto_slam

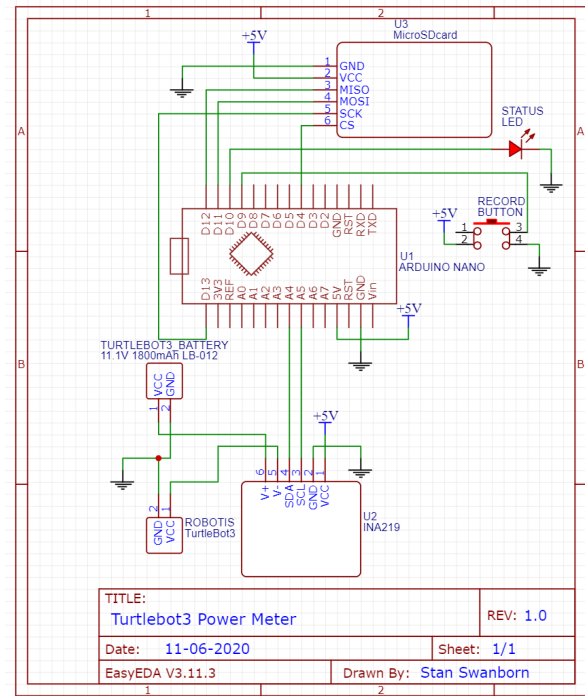


Figure 3. Electronic circuit schematic of the power meter/profiling device connecting an Arduino Nano board, an INA219 current sensor, and a MicroSD card module.

- **Robot Runner** [Swanborn and Malavolta, 2021] – A tool for automatic execution of measurement-based experiments on robotics software. Responsible for orchestration of the experiment by sending instructions to the robot from the remote PC.
- **Custom *ros1_ina219_profiler_service* package** – A custom ROS package used to start and stop profiling using the INA219 power sensor. Features (i) a client node that can invoke service calls to start and stop profiling, and (ii) a server node that runs on the system to be profiled. The client acts as a plugin for Robot Runner.
- **Custom *ros1_resource_profiler_service* package** – A custom ROS package used to start and stop CPU and memory profiling. Uses the *PSUTIL*¹³, which is the de-facto standard for system monitoring in Python. Features (i) a client node that can invoke service calls to start and stop profiling, and (ii) a server node that runs on the system to be profiled. As in the previous one, the client acts as a plugin for Robot Runner. Both of them are available in the replication package.
- **Rosbag CLI¹⁴** – Tool which facilitates the recording and playback of messages sent on all (or a subset of) ROS topics in a robotic system. Used for recording the trajectories for traversing the arenas in advance, as well as logging published messages during the experiment execution.

6.3 Environment and Mission Definition

This section describes the mission that the robot executes during the experiment as well as the environment in which it operates.

¹³<https://github.com/giampaolo/psutil>

¹⁴<http://wiki.ros.org/rosbag/Commandline>

6.3.1 Environment

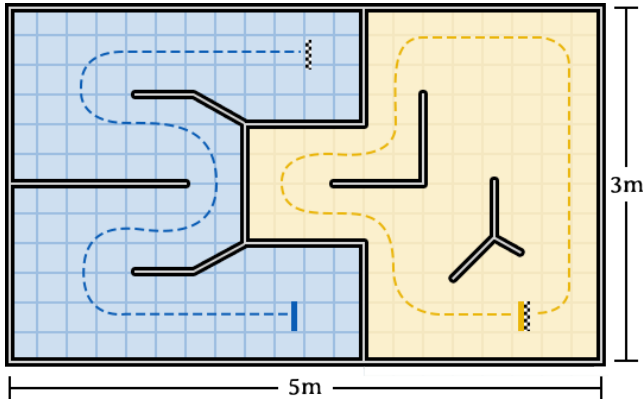


Figure 4. Design and approximate robot trajectory (dashed lines) for the point-to-point (blue area) and circular arenas (yellow area) within a 3x5 meters space.

Existing research is off-aligned regarding the environment in which an experiment involving SLAM is performed. Ranging from small empty rooms to university corridors, there appear to be no standard or well-defined guidelines regarding the experimental setup. In the absence of such, this study proposes the environment design as shown in Figure 4. It consists of two separate *arenas*, each with its own characteristics that may be relevant in the context of evaluating SLAM algorithms:

Point-to-point – Requires a trajectory with a distinct start and end point. The trajectory does not contain any *loop closures* (i.e., when traversing the arena, the robot never encounters any landmark it has seen before but was lost out of sight). Features three occasions where the robot makes a half turn, two straight sections (at the start and ending), and a single corridor. This arena is a simple polygon (i.e., does not intersect itself and has no holes).

Time to traverse: ± 80 seconds at 0.10 m/s

Circular – Offers a trajectory with an identical start and end point. Due to the large open space in the middle of the arena, the robot can look ahead to parts of the map it will traverse later on, thus loop closure can take place (i.e., familiar landmarks can be encountered multiple subsequent times). Features one occasion where the robot makes a half turn, six quarter turns, and two elongated straight sections (approximately 1.5 and 2 meters). This arena is *not* a simple polygon since it contains walls that are not connected to its outer boundaries.

Time to traverse: ± 90 seconds at 0.10 m/s

The dimensions of both arenas are chosen such that traversal is possible in a reasonable duration. A sequence of movement instructions for both trajectories depicted in Figure 4 is recorded using the *roslaunch* CLI tool. Playing back this recording enables autonomous traversal during the actual mission.

6.3.2 Mission definition

The mission for this experiment is executed fully on board the robot, without any interaction with the remote PC. It is orchestrated by a Python script with as input the current run's configuration for each factor in the experiment. The mission

script performs the following steps: i) initiates a *roslaunch*¹⁵ recording for all topics on the system; ii) launches the SLAM algorithm with the parameter configuration of the current run in a separate process; iii) idles for 5 seconds, allowing the SLAM routine to initialize itself; iv) traverses the arena by playing back the prerecorded movement instructions; v) upon completion of traversal, terminates the SLAM process; vi) stops the *roslaunch* recording.

Recording a *roslaunch* while executing the mission enables the extraction of metrics related to ROS messages as introduced in Section 4.1 and may aid in the explanation of observations during data analysis. Furthermore, the set of all 320 recorded *roslaunch*s obtained during the experiment may be useful for future research.

Throughout the entire experiment, the only configuration parameters of the SLAM algorithm that change are those that are listed as experiment factors in Section 5.4. All other parameters are kept at their default values, with two exceptions:

- Parameters related to the *laser sensor* (e.g., maximum and minimum range) are configured according to the specification of the manufacturer¹⁶. This ensures the correct operation of the sensor.
- The *temporal update* parameter of the Gmapping algorithm is disabled for all the runs, despite it being enabled in the default configuration of the algorithm. Enabling this parameter would trigger the processing of new laser scans after a given period has passed, thereby possibly bypassing the linear and angular update thresholds that are set during the experiment. This could result in polluted measurements and is therefore undesirable.

6.4 Running the Experiment

The execution of this experiment relies on the Robot Runner tool as introduced by Swanborn and Malavolta [2021]. Robot Runner offers a framework for orchestrating and (partially) automating the execution of an experiment involving robotics software. Designed with a particular focus on runtime properties, it is a suitable vehicle for executing the experiment in the present study.

The experiment execution is structured through Robot Runner's *event manager*, which facilitates the invocation of user-defined callbacks to key events in the experiment lifecycle. The subset of available events that is used in this experiment as well as their order of occurrence is depicted in Figure 5.

The procedure for each event is as follows:

1. **before_run** – Show which arena will be used for the next run and prompt the experiment operator to place the robot there.

¹⁵<http://wiki.ros.org/roslaunch>

¹⁶http://wiki.ros.org/turtlebot3_slam

Manual intervention

Before each experiment run, the operator equips the robot with a fully charged battery. While the TurtleBot's battery could last more than one run without a problem, using a fully charged battery for each run eliminates the risk of throttling or declined performance as a result of diminishing capacity. Only after the operator presses the **enter** key to signify the correct placement of the robot in the marked starting position for the correct arena, does the experiment proceed.

2. **start_run** – Initialise a new ROS node as well as the energy and resource profiler clients that rely on it.
3. **start_measurement** – Invoke a service call through the energy and resource profiler clients to start profiling.
4. **launch_mission** – Open an SSH connection with the robot and invoke the command to start the mission. The mission then operates fully autonomously on the robot itself. While the SSH connection persists, **no** communication takes place between the remote pc and the robot until the mission terminates. All terminal output is logged to disk instead, for subsequent retrieval.
5. **stop_measurement** – Invoke a service call through the energy and resource profiler clients to stop profiling. The measurements are obtained along with the response from the profiler server (running on the robot) and written to their respective CSV files in the current run's directory.
6. **stop_run** – Copy all mission logs and the bag containing all recorded ROS messages from the robot to the current run's directory over an SSH connection.
7. **populate_run_data** – Augment the row in the run table corresponding to the current run with new values for average CPU utilization, memory, and energy measurements. To avoid including any excess measurements in the analysis, aggregation takes place only for those values whose timestamps lie between the start of the rose bag recorded during the mission and 1 second after the last movement command with a non-zero velocity. Measurements outside of this period may result from e.g., network latency during the service calls to start and stop profiling, and are unrelated to the mission itself.

The above sequence of steps is repeated for all remaining runs in the experiment. After each run is completed, a cool-down period of 60 seconds takes effect before proceeding with the next run. This is done to stabilize the robot's conditions (such as CPU load and battery consumption).

7 Results

This section presents the results obtained from the experiment. The interpretation and analysis of the data is performed according to the procedure defined in Section 5.5. The graphs and test results included in this section reflect a *summary* of the analysis as a whole, extracted from the larger body of all tables and figures produced. The full range of R and Python scripts used during analysis as well as all complete results are available in the replication package of this study.

7.1 Energy Consumption

This section presents the results related to **RQ1.1**: “How do SLAM algorithms impact energy consumption of ROS-based systems?”

7.1.1 Exploration

Table 6 describes energy measurement statistics grouped per algorithm. It is possible to observe that *Gmapping* is simultaneously associated with the maximum and minimum of all energy consumption measurements for both, the point-to-point and circular arena. The previous observation becomes even more apparent in the histograms pictured in Figure 7b and Figure 6b. Considering its standard deviation and distribution of quantiles in comparison with the other algorithms, *Gmapping* exhibits the most variation in energy consumption across the board.

Visual inspection makes it possible to observe that the energy measurements for most algorithms are close to each other, whereas measurements for *Gmapping* appear to follow a more dichotomous distribution. This could be an indicator of another factor affecting energy consumption for *Gmapping* in particular, which does not have the same effect on the other algorithms. This suspicion is further supported by the box plots in Figure 7a when comparing *Gmapping* boxes between low and high values for the angular update threshold. While other algorithms are less affected by this parameter change, *Gmapping*'s average energy consumption is up to 38 Joules lower for the high angular update threshold. A similar situation is observed for the point-to-point arena, as shown in Figure 6b and Figure 6a.

A peculiarity in Figure 7a is the extreme range of the middle two quartiles indicated by the *Gmapping* box for the low linear and angular update thresholds at a map resolution of 0.05 m/s. Further inspection of the data reveals that this is due to runs 9 and 10, both of which resulted in exceptionally low energy consumption. These runs correspond to the far low end of the histogram in Figure 7b. Visual inspection of the maps produced by these runs and checking of

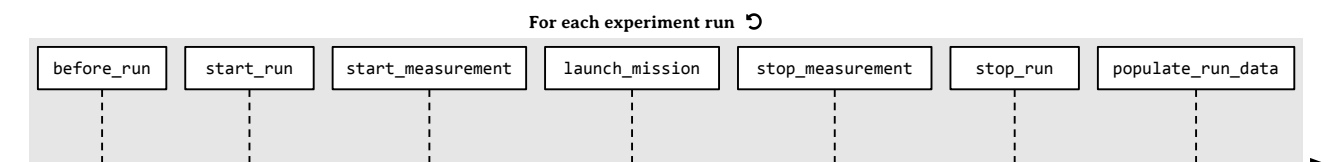


Figure 5. Timeline of subscribed experiment events from the Robot Runner framework [Svanborn and Malavolta, 2021].

Table 6. Overview of energy consumption (J) per algorithm.

Arena	Algorithm	Minimum	1st quartile	Median	Mean	Stddev.	3rd quartile	Maximum
Point to point	Cartographer	804.622	822.628	825.942	824.631	6.582	827.950	838.958
	Gmapping	787.603	810.210	818.595	824.167	17.160	839.843	855.046
	Hector	799.365	805.203	807.303	807.419	3.863	809.756	821.977
	Karto	789.650	805.044	807.148	807.344	5.203	810.330	823.203
Circular	Cartographer	945.967	951.226	956.550	955.367	4.617	958.795	962.301
	Gmapping	903.713	938.833	945.841	953.369	20.573	972.539	986.023
	Hector	925.266	930.026	933.742	934.337	5.270	936.313	945.779
	Karto	921.365	929.733	932.745	932.841	4.968	936.152	944.203

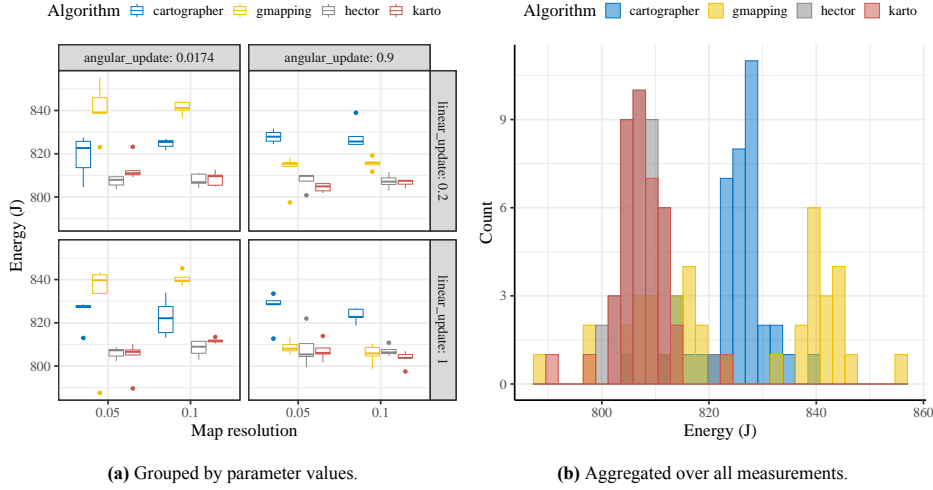


Figure 6. Energy consumption for the point-to-point arena. The histograms in Figure 6b depict Gmapping as having the highest variability in energy consumption. This finding aligns with the plot in Figure 6a, where Gmapping is the most energy-intensive in the first and third quarters, shows a median level of consumption in the second quarter, and is among the least expensive in the final quarter.

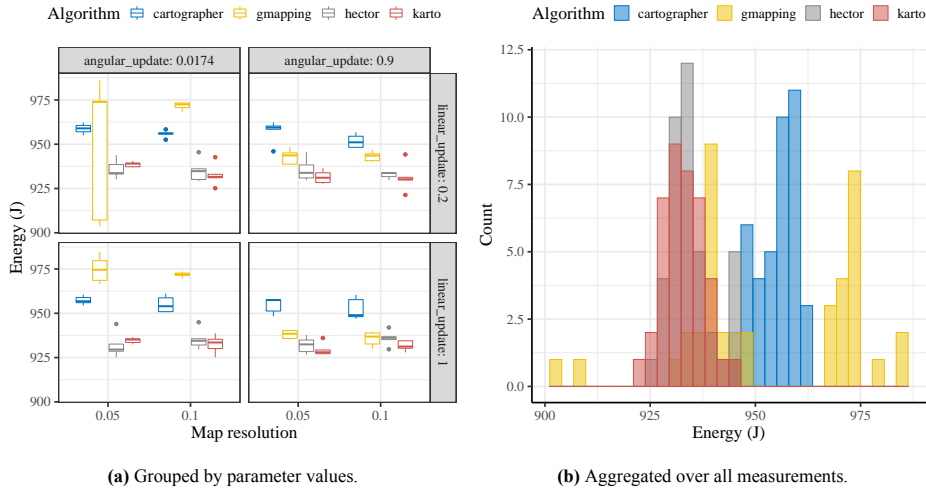


Figure 7. Energy consumption for the circular arena. Compared to Figure 6, Figure 7b reveals a slightly different distribution of measurement values and shows an anomalous variation in the Gmapping boxplot during the first quarter. Despite these differences, all algorithms exhibit a similar level of resource consumption as in the point-to-point arena, as seen in Figure 6, with Gmapping again demonstrating the widest variability.

the error logs revealed no irregularities, so there is no concrete evidence suggesting measurement errors. While these runs may seem anomalous, we opted to retain them in the analysis. This decision aligns with ethical data analysis practices, where excluding data without a solid basis can introduce bias. Moreover, removing outliers in the absence of strong evidence is generally discouraged, as it risks distorting the results. In this case, the overall trends observed in the data remain consistent with or without the inclusion of these runs, and the main conclusions drawn from the study are not

substantially affected by these outliers.

Visual inspection of the time series depicted in Figure 8 again reveals a different pattern for Gmapping compared to the other algorithms. The patterns repeat in both arenas, where the power consumption of Gmapping seems to have an apparent periodic component, likely caused by the processing of new laser scans (triggered by movement). Further inspection of the power consumption of individual Gmapping runs supports this, showing periodic spikes in power consumption for those stretches of the trajectory where the robot

moves straight forward. The average power consumption increases even further once the robot traverses a bend. This is also clearly visible in Figure 8, with the first bend occurring around 13.5 and 22.7 seconds into the run, respectively. The low angular update setting seems to result in an especially large increase in power consumption at points where the trajectory bends.

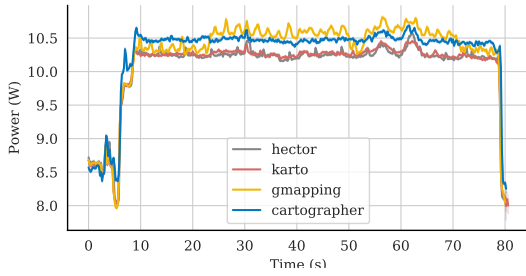


Figure 8. The power consumption over time for single runs using different SLAM algorithms in the point-to-point arena reveals distinct patterns. The graph indicates that Gmapping exhibits periodic fluctuations in power consumption, particularly during the intervals of 10-20, 20-50, and 55-65 seconds, in contrast to the other algorithms.

The other algorithms show only minor deviations in power consumption throughout the majority of the run. The most obvious changes in power consumption correspond with the starting and stopping of the actuators that move the robot.

7.1.2 Statistical analysis – point to point arena

As a first step in the statistical analysis, the ANOVA assumptions were verified for the measurements to be analyzed. Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data suggested a deviation from normality. This was confirmed by a significant Shapiro-Wilk test of normality ($p = 2.362698e - 12$), leading to the rejection of the null hypothesis that the residuals follow a normal distribution. Several transformations, including logarithm, square root, and reciprocal, were applied to address this issue. While these transformations adjusted the distribution, they did not sufficiently correct their normality. Furthermore, while reducing skewness, such transformations also impact the interpretability of the results. For instance, log transformation compresses larger values more than smaller ones, changing the relationship between variables, and consequently, affecting the practical interpretation of effect sizes. Similarly, the square root and reciprocal transformations change the scale of the data (with a lower impact on the difference among data points), which challenges the interpretation of the original measurement scales.

Given that normality was not achieved through these transformations, we opted for PERMANOVA, which does not rely on normality assumptions. Levene’s test for homogeneity of variances was insignificant ($p = 0.545$), supporting the assumption of equal variances across groups. To enhance the robustness of the PERMANOVA analysis and mitigate the influence of randomness, we used 100 million random permutations, balancing computational feasibility with statistical rigor.

PERMANOVA results indicate a significant main effect for the algorithm ($p < 2e^{-16}$), linear update threshold ($p =$

0.02697), and angular update threshold factors ($p < 2e^{-16}$). A significant two-way interaction was found between the algorithm and angular update threshold ($p < 2e^{-16}$). Since there are only two treatments for the linear update factor (high and low), a post-hoc test was not required for this main effect. The difference between the high and low linear update thresholds is minimal, with average energy consumption of 814.826 J and 816.954 J, respectively, a relative increase of just 0.3%.

Post-hoc analysis of the main effect of the algorithm-angular update interaction proceeds with normality assessment. 4 out of 8 combinations of the algorithm and angular update factors yield a significant p-value, substantiated by non-normal Q-Q plots. Transforming the data does not lead to insignificance, hence the assumptions for Tukey’s HSD test are not met. The result of Levene’s test is not significant ($p = 0.2133176$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. Proceeding with Dunn’s test is thus appropriate.

Dunn’s test with Benjamini-Hochberg correction showed no significant differences between any combination of Hector and Karto (i.e., for both high and low angular update thresholds). This result is intuitive given the visual proximity of their distributions in Figure 6b. Significant differences between the low and high angular update threshold were found within Gmapping ($p = 2.5e^{-06}$) and Karto ($p = 0.04980$) itself. As a consequence of this, a dichotomy arises: All comparisons between the Cartographer and the other groups yield a significant difference, except for Gmapping with a low angular update threshold.

Result: Energy consumption - Point to point arena

A lower angular update threshold results in a statistically significant increase in energy consumption within:

Gmapping: 810.523 J to 837.811 J (+3.4%);

Karto: 805.392 J to 809.296 J (+0.5%).

Considering the algorithm – angular update interaction, two groups can be identified:

- Measurements for **Gmapping at a high angular update threshold**, **Hector** and **Karto** are not significantly different. Their combined average energy consumption is 808.0096J;
- Measurements for **Gmapping at a low angular update threshold** and **Cartographer** are not significantly different. Their combined average energy consumption is 829.0247 J (+2.6%).

7.1.3 Statistical analysis – Circular arena

Moving on to the circular arena, again the ANOVA assumptions are verified first. Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 5.964346e^{-16}$). Thus, the null hypothesis that the residuals originate from a normal distribution is rejected. This is still the case after transforming the data. Furthermore, Levene’s test is significant ($p = 0.0003465$), so its null hypothesis is rejected and

homogeneity of variances may not be assumed. Since the assumptions for ANOVA are violated, the analysis resorts to PERMANOVA (again with a limit of 100 million random permutations). The results indicate a significant main effect of the algorithm ($p < 2e^{-16}$) and angular update threshold factors ($p < 2e^{-16}$). Furthermore, a significant two-way interaction between the algorithm and angular update ($p < 2e^{-16}$) as well as a three-way interaction between the algorithm, linear update and angular update are found ($p = 0.02355$). To investigate the three-way interaction, analysis is split across both levels of the linear update factor.

Low linear update threshold – Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 2.967038e^{-11}$). Thus, the null hypothesis that the residuals originate from a normal distribution is rejected. This is still the case after transforming the data. Furthermore, Levene’s test is significant ($p = 0.0041560$), so its null hypothesis is rejected and homogeneity of variances may not be assumed. Since the assumptions for ANOVA are violated, the analysis resorts to PERMANOVA (100 million random permutations). Results indicate a significant main effect of the algorithm ($p < 2e^{-16}$) and angular update threshold factors ($p = 0.01410$). Since there are only two treatments for the angular update factor (high and low) a post-hoc test for this main effect is not required.

Post-hoc analysis of the main effect of the algorithm proceeds with normality assessment. 2 out of 4 algorithms (Hector and Gmapping) yield a significant p-value, substantiated by non-normal Q-Q plots. Transforming the data does not lead to insignificance, hence the assumptions for Tukey’s HSD test are not met. Consequently, the non-parametric Dunn’s test with Benjamini-Hochberg correction is performed. The results show no significant differences between Hector and Karto, nor between Cartographer and Gmapping. However, the difference between any two members across these groups is significant. This result is again reflected in the visual proximity of the distributions in the top half of Figure 7a.

Result: Energy consumption - Circular arena

For the low linear update threshold, a lower angular update threshold results in a statistically significant increase in energy consumption: from 940.761 J to 947.202 J (+0.7%).

Two groups of algorithms can be identified:

- Measurements for **Hector** and **Karto** are not significantly different. Their combined average energy consumption is 934.195J;
- Measurements for **Gmapping** and **Cartographer** are not significantly different. Their combined average energy consumption is 953.7685 J (+2.1%).

High linear update threshold – Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data seems to indicate a normal distribution. This is confirmed by a non-significant Shapiro-Wilk test of normality

($p = 0.3506791$). Furthermore, Levene’s test is not significant ($p = 0.6505165$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. This means that all assumptions for the regular ANOVA are met. ANOVA Results indicate a significant main effect of the algorithm ($p < 2e^{-16}$) and angular update threshold factors ($p = 8.22e^{-15}$). Furthermore, a significant two-way interaction between the algorithm and angular update are found ($p < 2e^{-16}$).

Post-hoc analysis of the algorithm-angular update interaction proceeds with normality assessment. All 8 combinations of the algorithm and angular update factors yield a non-significant p-value, substantiated by seemingly normal Q-Q plots. The result of Levene’s test is not significant ($p = 0.5175312$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. The assumptions for Tukey’s HSD test are met. Results with p-values are corrected for multiple comparisons and show no significant differences between any combination of Hector and Karto (i.e., for both high and low angular update thresholds). This result is intuitive given the visual proximity of their distributions in Figure 7a. Significant differences between the low and high angular update threshold were found within Gmapping itself: the lower angular update threshold corresponds to higher energy consumption. Similar to the point-to-point arena, all comparisons between Cartographer and the other groups yield a significant difference, except for Gmapping with a low angular update threshold.

Result: Energy consumption - Circular arena

For the high linear update threshold, a lower angular update threshold results in a statistically significant increase in energy consumption within:

Gmapping: 936.823 J to 973.357 J (+3.9%).

Considering the algorithm – angular update interaction, two groups can be identified:

- Measurements for **Gmapping at a high angular update threshold**, **Hector** and **Karto** are not significantly different. Their combined average energy consumption is 933.751J;
- Measurements for **Gmapping at a low angular update threshold** and **Cartographer** are not significantly different. Their combined average energy consumption is 961.0167 J (+2.9%).

7.2 CPU Utilization

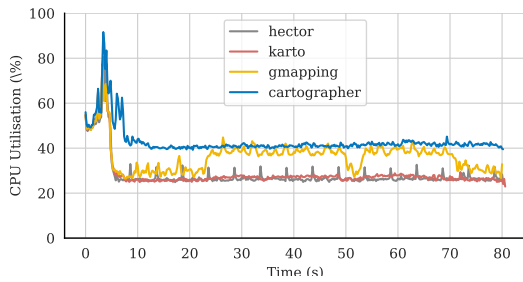
This section presents the results related to **RQ1.2**: “How do SLAM algorithms impact CPU utilization of ROS-based systems?”

7.2.1 Exploration

An overview of descriptive statistics for the (average) CPU utilization grouped per algorithm is provided in Table 7. The difference in mean CPU utilization between the most and least demanding algorithms is similar for both arenas: Compared to Hector, the CPU utilization of Cartographer is

Table 7. Overview of CPU utilisation (%) per algorithm.

Arena	Algorithm	Minimum	1st quartile	Median	Mean	Stddev.	3rd quartile	Maximum
Point to point	Cartographer	37.193	41.256	42.841	43.102	2.522	43.957	49.586
	Gmapping	28.055	31.218	38.939	36.816	6.293	40.917	55.445
	Hector	27.039	27.596	28.414	28.879	1.434	30.492	31.465
	Karto	26.414	27.848	28.880	29.145	1.953	30.417	34.661
Circular	Cartographer	39.629	42.071	42.619	42.958	1.572	43.996	46.025
	Gmapping	28.253	31.621	37.771	38.074	7.483	43.273	57.691
	Hector	27.175	27.662	28.316	28.605	1.149	29.359	31.471
	Karto	26.608	28.364	29.246	29.343	1.716	30.189	32.974

**Figure 9.** CPU utilization over time for single runs using different SLAM algorithms in the point-to-point arena. Gmapping reveals again a different pattern compared to the other algorithm, and Cartographer is the one that keeps CPU usage higher over time.

14.2% and 14.4% higher for the point-to-point and circular arena, respectively. This comes down to a relative increase of up to 50.1% in processing power. Similar to the energy measurements in Section 7.1, Gmapping exhibits the most variation in CPU utilization when observing the standard deviation and distribution of quartiles.

Using visual inspection of Figure 10, it seems that the CPU utilization measurements are mostly clustered per algorithm. Gmapping measurements appear to follow a more dichotomous distribution, just as for the energy measurements. Indeed, the angular update factor may affect the CPU utilization for Gmapping in particular, providing a potential cause for the previously discovered differences in energy usage. This suspicion is supported by the box plots in Figure 10a and Figure 10b when comparing the Gmapping boxes between low and high values for the angular update threshold. Whereas other algorithms do not seem to be heavily affected by this parameter change, the average CPU utilization for Gmapping is up to 12% lower for the high angular update threshold.

Visual inspection of the time series depicted in Figure 9 reveals again a different pattern for Gmapping compared to the other algorithms. For both arenas, the CPU utilization increases by up to 15% at points where the trajectory bends (from 20–30 to 70–80, simultaneous with the increase in energy consumption as described in Section 7.1.1).

The other algorithms show only minor deviations in CPU utilization throughout the majority of the run. The most obvious changes in power consumption correspond with the moment algorithms are first launched at the start of the run. This is associated with a high CPU utilization for all algorithms and in both arenas.

7.2.2 Statistical analysis – point to point arena

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 4.629686e^{-9}$). Thus, the null hypothesis that the residuals originate from a normal distribution is rejected. This is still the case after transforming the data. Levene’s test is not significant ($p = 0.8954898$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. However, since the assumptions for ANOVA are violated, the analysis resorts to PERMANOVA (100 million random permutations). The results indicate a significant main effect of the algorithm ($p < 2e^{-16}$), map resolution ($p = 0.00818$), and angular update threshold factors ($p < 2e^{-16}$). Furthermore, a significant two-way interaction between the algorithm and angular update emerges ($p < 2e^{-16}$). Since there are only two treatments for the linear update factor (high and low) a post-hoc test for this main effect is not required.

Post-hoc analysis of the main effect of the algorithm – angular update interaction proceeds with normality assessment. 6 out of 8 combinations of the algorithm and angular update factors yield a significant p-value, substantiated by non-normal Q-Q plots. The assumptions for Tukey’s HSD test are therefore not met. The result of Levene’s test is not significant ($p = 0.2213964$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed.

Dunn’s test with Benjamini-Hochberg correction showed no significant differences between any combination of Hector and Karto (i.e., for both high and low angular update thresholds). Significant differences between the low and high angular update thresholds were found within Gmapping ($p = 0.00881$): the lower angular update threshold corresponds to higher CPU utilization. Similar to the energy measurements, all comparisons between the Cartographer and the other groups yield a significant difference, except for Gmapping with a low angular update threshold.

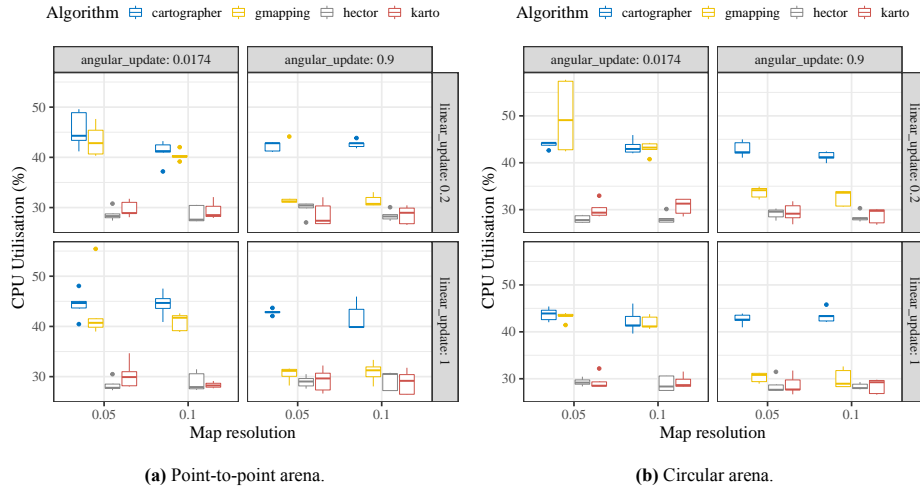


Figure 10. CPU utilization grouped by parameter values for both arenas. Despite Gmapping being the most susceptible to changes of parameter values in both arenas, Cartografer results in higher CPU utilization in all the configurations.

Result: CPU utilization - Point to point arena

On average, a higher map resolution (0.05 m/pixel vs 0.10 m/pixel) resulted in a statistically significant increase in CPU utilization: 33.990% to 34.981% (+0.99%, relative difference +2.9%).

On average, a lower angular update threshold resulted in a statistically significant increase in CPU utilization within: **Gmapping**: 31.651% to 41.982% (+10.3%, relative difference +32.6%).

Considering the algorithm – angular update interaction, two groups can be identified:

- Measurements for **Gmapping at a high angular update threshold**, **Hector** and **Karto** are not significantly different. Their combined average CPU utilization is 29.5398%;
- Measurements for **Gmapping at a low angular update threshold** and **Cartographer** are not significantly different. Their combined average CPU utilization is 42.7283% (+13.2%, relative difference +44.6%).

resolution factor (high and low) a post-hoc test for this main effect is not required.

In a Shapiro-Wilk test 4 out of 8 combinations of the algorithm and angular update factors yield a significant p-value, substantiated by non-normal Q-Q plots. The assumptions for Tukey's HSD test are therefore not met. Dunn's test with Benjamini-Hochberg correction showed no significant differences between any combination of Hector and Karto (i.e., for both high and low angular update thresholds). Significant differences between the low and high angular update thresholds were found within Gmapping ($p = 0.00039$): the lower angular update threshold corresponds to higher CPU utilization. Similar to the energy measurements, all comparisons between the Cartographer and the other groups yield a significant difference, except for Gmapping with a low angular update threshold.

7.2.3 Statistical analysis – Circular arena

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 2.544732e^{-9}$). After transforming the measurement data with a reciprocal transformation, the data seems normally distributed. This is confirmed by a non-significant Shapiro-Wilk test of normality ($p = 0.1779216$). For the transformed data, Levene's test is not significant either ($p = 0.4157229$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. ANOVA results indicate a significant main effect of the algorithm ($p < 2e^{-16}$), map resolution ($p = 0.03133$), linear update ($p = 0.00562$), and angular update threshold factors ($p < 2e^{-16}$). Furthermore, a significant two-way interaction between the algorithm and linear update ($p = 0.00030$) as well as the algorithm and angular update ($p < 2e^{-16}$) emerges. Since there are only two treatments for the map

2 out of 8 combinations of the algorithm and linear update factors yield a significant p-value in a Shapiro-Wilk test. Furthermore, Levene's normality test is significant ($p = 3.717794e^{-30}$). The assumptions for Tukey's HSD test are not met. Dunn's test with Benjamini-Hochberg correction showed no significant differences between any combination of Hector and Karto (i.e., for both high and low linear update thresholds). Additionally, no significant differences between the low and high linear update thresholds were found within any algorithm itself. All comparisons between Cartographer and the other algorithms yield a significant difference, except for Gmapping with a high linear update threshold. This is counterintuitive since the two different linear threshold levels for Gmapping showed no significant difference with one another. It is expected that the potential outlier runs 9 and 10 (Section 7.1.1) are the cause of this: they both have exceptionally high CPU utilization, contributing to the large standard deviation within the Gmapping measurements that can be seen in Table 7.

Table 8. Overview of memory utilization (MB) per algorithm.

Arena	Algorithm	Minimum	1st quartile	Median	Mean	Stddev.	3rd quartile	Maximum
Point to point	Cartographer	653.905	657.044	657.562	657.683	1.567	658.830	660.205
	Gmapping	616.649	622.972	628.039	630.554	11.411	631.600	663.677
	Hector	737.314	743.926	745.477	745.021	3.086	746.887	751.010
	Karto	615.261	621.227	622.781	622.740	2.524	624.428	626.585
Circular	Cartographer	651.944	657.513	658.857	659.147	2.681	660.856	663.986
	Gmapping	621.964	625.816	627.805	629.148	6.868	631.219	663.956
	Hector	742.550	746.236	748.937	748.499	3.631	750.836	757.783
	Karto	615.764	621.314	623.551	623.073	2.966	625.114	628.447

Result: CPU utilisation - Circular arena

On average, a higher map resolution (0.05 m/pixel vs 0.10 m/pixel) resulted in a statistically significant increase in CPU utilization: 34.316% to 35.173% (+0.86%, relative difference +2.5%).

On average, a lower angular update threshold resulted in a statistically significant increase in CPU utilization within:

Gmapping: 31.636% to 44.511% (+12.9%, relative difference +40.7%).

Considering the algorithm – angular update interaction, two groups can be identified:

- Measurements for **Gmapping at a high angular update threshold**, **Hector** and **Karto** are not significantly different. Their combined average CPU utilization is 29.506%;
- Measurements for **Gmapping at a low angular update threshold** and **Cartographer** are not significantly different. Their combined average CPU utilization is 43.4757% (+14.0%, relative difference +47.3%).

7.3 Memory Utilisation

This section presents the results related to **RQ1.3:** “How do SLAM algorithms impact memory utilization of ROS-based systems?”

7.3.1 Exploration

An overview of descriptive statistics for the (average) memory utilization grouped per algorithm is provided in Table 8. An observation from the means in this table, reveals that Hector utilized the most memory for both arenas. The difference with other algorithms ranges up to 125MB for Karto, the most memory-efficient algorithm of the four.

Given the order of magnitude of the measurements in Table 8, the standard deviation across all algorithms is relatively low. This gives the impression that the parameter configuration of the algorithms does not have a major impact on memory utilization. This is also visible in Figure 11, showing tight boxes that appear largely clustered by the algorithm. Gmapping and Karto appear close together on the lower end of the spectrum, followed by cartographers at a visually distinct peak. As could be derived from Table 8, it is easily visible that the memory utilization for Hector is the largest.

Considering the time series data depicted in Figure 12, it becomes visible that the memory utilization of most al-

gorithms increases only marginally throughout the mapping process. An exception to this rule is Hector, which not only starts with the highest utilization but also shows the highest increase in memory utilization as time progresses.

7.3.2 Statistical analysis – Point to point

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 2.690573e^{-13}$). Levene’s test is significant as well ($p = 0.01517715$), so its null hypothesis is rejected and homogeneity of variances may not be assumed. ANOVA assumptions are thus violated. PERMANOVA results indicate a significant main effect of the algorithm ($p < 2e^{-16}$) and map resolution factors ($p = 0.005977$). Furthermore, a significant two-way interaction between the algorithm and map resolution ($p = 3.5e^{-7}$) emerges.

In a Shapiro-Wilk test 4 out of 8 combinations of the algorithm and map resolution factors yield a significant p-value, substantiated by non-normal Q-Q plots. The assumptions for Tukey’s HSD test are therefore not met. Dunn’s test with Benjamini-Hochberg correction showed no significant difference in memory utilization between both map resolutions within any of the algorithms. All algorithms show significant differences in memory utilization when compared with one another, except Gmapping at a low map resolution which is not significantly different from both resolutions in Karto.

7.3.3 Statistical analysis – Circular

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 3.981659e^{-13}$). Levene’s test is not significant ($p = 0.01517715$), so its null hypothesis cannot be rejected and homogeneity of variances may be assumed. However, non-normality of residuals means that not all assumptions for ANOVA are met. PERMANOVA results indicate a significant main effect of the algorithm ($p < 2.2e^{-16}$) and angular update factors ($p = 0.001313$). Furthermore, a significant two-way interaction between the algorithm and map resolution ($p = 4.612e^{-5}$) emerges. Since there are only two treatments for the angular update factor (high and low) a post-hoc test for this main effect is not required. The observed difference in memory utilization between these treatments (663.909 MB and 666.025 MB respectively, +0.3%) is minor.

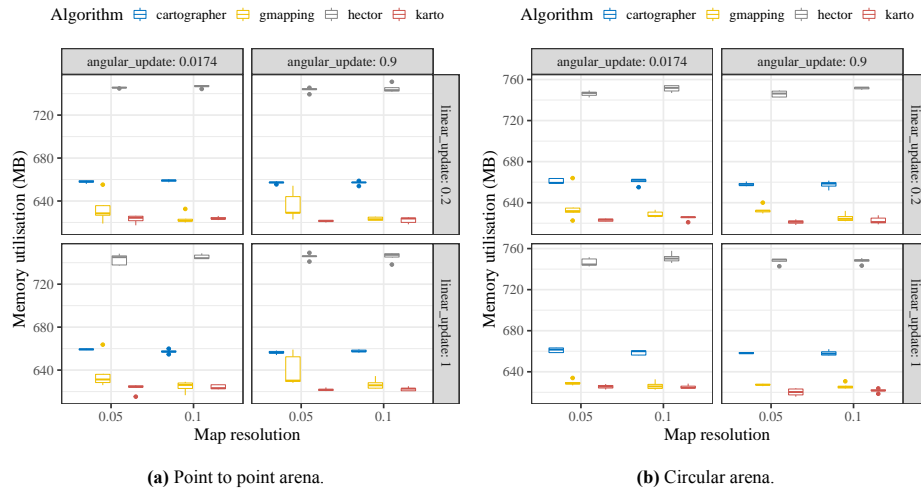


Figure 11. Memory utilization grouped by parameter values. The tight boxes appear largely clustered by the algorithm giving the impression that the parameter configuration of the algorithms does not have a major impact on memory utilization. Gmapping and Karto appear close together on the lower end of the spectrum, followed by cartographers at a visually distinct peak, while Hector is the one that results in more memory usage.

In a Shapiro-Wilk test 3 out of 8 combinations of the algorithm and angular update factors yield a significant p-value, substantiated by non-normal Q-Q plots. The assumptions for Tukey's HSD test are therefore not met. The results of Dunn's test with Benjamini-Hochberg are identical to what has been reported for the point-to-point arena: no significant difference in memory utilization could be found between map resolutions within any of the algorithms. All algorithms show significant differences in memory utilization when compared with one another, except Gmapping at a low map resolution which is not significantly different from both resolutions in Karto.

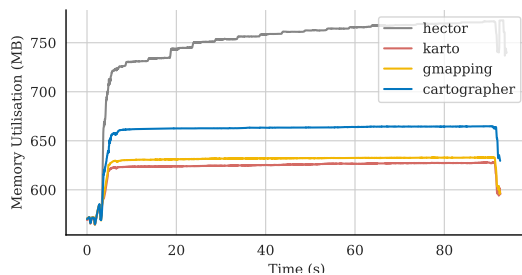


Figure 12. Memory utilization over time for the circular arena. The memory utilization of most algorithms increases only marginally throughout the mapping process, with Hector as an exception, which not only starts with the highest utilization but also shows the highest increase in memory utilization as time progresses.

Result: Memory utilisation - Point to point arena

The memory utilization of all algorithms increases over time. Comparing the average memory consumption between the start and end of the robot's trajectory:

Karto: 623.5126 MB to 627.2033 MB (+0.6%);

Gmapping: 629.6335 MB to 635.7442 MB (+1.0%);

Cartographer: 652.1205 MB to 663.7730 MB (+1.8%);

Hector: 723.1475 MB to 770.3595 MB (+6.5%).

On average, different map resolutions caused no statistically significant difference in memory utilization for any of the algorithms.

Considering the algorithm – map resolution interaction, four groups can be identified:

- Measurements for **Gmapping at a low map resolution** and **Karto** are not significantly different. Their combined average memory utilization is 623.3813 MB;
- Measurements for **Gmapping at a high map resolution** are not significantly different from **Gmapping at a low map resolution**, though the former is significantly different from Karto. Their combined average memory utilization is 630.554 MB (+1.2%).
- The average memory utilization for **Cartographer** is 657.683 MB (+5.5%).
- The average memory utilization for **Hector** is 745.0205 MB (+19.5%).

Result: Memory utilisation - Circular arena

The memory utilization of all algorithms increases over time. Comparing the average memory consumption between the start and end of the robot's trajectory:

Karto: 622.1176 MB to 627.1456 MB (+0.8%);

Gmapping: 628.5837 MB to 632.9714 MB (+0.7%);

Cartographer: 657.5676 MB to 664.2516 MB (+1.0%);

Hector: 722.8760 MB to 771.8021 MB (+6.8%).

On average, different map resolutions caused no statistically significant difference in memory utilization for any of the algorithms.

Considering the algorithm – map resolution interaction, four groups can be identified:

- Measurements for **Gmapping at a low map resolution** and **Karto** are not significantly different. Their combined average memory utilization is 624.271 MB;
- Measurements for **Gmapping at a high map resolution** are not significantly different from **Gmapping at a low map resolution**, though the former is significantly different from Karto. Their combined average memory utilization is 629.1475 MB (+0.8%).
- The average memory utilization for **Cartographer** is 659.1475 MB (+5.6%).
- The average memory utilization for **Hector** is 748.499 MB (+19.9%).

7.4 ROS Messages

This section presents the results related to **RQ1.4**: “How do SLAM algorithms impact message traffic in ROS-based systems?”

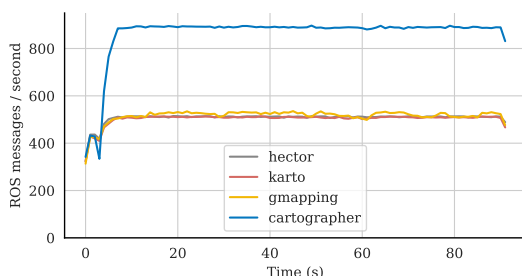


Figure 13. Message rate over time for the circular arena. We observe a high discrepancy of Cartographer. Once the robot starts driving and new areas are mapped, there is a huge increase in the number of ROS messages.

7.4.1 Exploration

An analysis of the message count per run compared to the total message size using Spearman's correlation coefficient indicated a strong correlation between the two ($\rho = 0.999212$). Tables and figures for this metric have therefore been omitted from this section to save space. The results and subsequent analysis presented for the message count also hold for the message size metric (the full analysis is available in the replication package of this study). Table 9 presents an overview of descriptive statistics for the message count per run. From

the means in this table, it is possible to observe that Cartographer is associated with the highest number of ROS messages per run for both arenas. On average, Cartographer message counts are up to 70% higher than for the other algorithms.

Considering the large order of magnitude of the measurements in Table 9 in combination with relatively low standard deviations across all algorithms, the parameter configuration of the algorithms does not seem to have a major impact on ROS message count. This is also visible in Figure 14, showing tight boxes and histograms that likewise for memory usage, appear largely clustered by the algorithm. Again, as became apparent from Table 9, the message count for Cartographer is larger than for the other algorithms.

The discrepancy of Cartographer is also visible in Figure 13. In the figure, once the robot starts driving and new areas are mapped, there is a huge increase in the number of ROS messages. From this point onward the rate remains stationary for the remainder of the run.

Further investigation of the ROS topics that are published during the run reveals the reason for the difference in message counts between the algorithms. The Cartographer package in particular uses a broad range of additional topics for communication between nodes that comprise the SLAM system, such as `flat_imu` and `costmap_updates`.

7.4.2 Statistical analysis – Point to point

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 1.303741 \cdot 10^{-15}$). Levene's test is not significant ($p = 0.2348712$), so its null hypothesis cannot be rejected. While homogeneity of variances may thus be assumed, the ANOVA assumptions are still violated. PERMANOVA results indicate a single significant main effect of the algorithm ($p < 2e^{-16}$). Post-hoc analysis shall show which pairs of algorithms are significantly different.

In a Shapiro-Wilk test, all 4 algorithms yield a significant p-value, substantiated by non-normal Q-Q plots. Thus the assumptions for Tukey's HSD test are violated. Dunn's test with Benjamini-Hochberg correction indicates a significant difference in message count between all pairs of algorithms except Hector-Karto.

Result: ROS message count - Point to point arena

From the four algorithms, three groups can be identified:

- Measurements for **Hector** and **Karto** are not significantly different. Their combined average message count is 40367.81;
- The average message count for **Gmapping** is 41198.32 (+2.1%);
- The average message count for **Cartographer** is 68170.48 (+68.9%).

7.4.3 Statistical analysis – Circular

Visual inspection of the Q-Q plot of residuals from a linear model fitted to the data does not seem to indicate a normal

Table 9. Overview of message count per algorithm

Arena	Algorithm	Minimum	1st quartile	Median	Mean	Stddev.	3rd quartile	Maximum
Point to point	Cartographer	61278	67722.50	69229.0	68170.48	2268.840	69358.75	69754
	Gmapping	30036	40540.75	41387.5	41198.32	2700.456	43034.25	44426
	Hector	39376	40441.50	40708.0	40579.65	384.277	40835.50	41226
	Karto	34553	40160.25	40465.5	40155.97	1073.233	40599.00	40868
Circular	Cartographer	74940	79850.5	80100.0	79666.23	1258.845	80337.50	80725
	Gmapping	33937	47053.5	47324.5	47430.38	3429.072	49566.00	51093
	Hector	46132	46549.0	46704.5	46746.65	317.821	46972.25	47404
	Karto	44027	46137.5	46627.5	46359.22	759.921	46837.00	47319

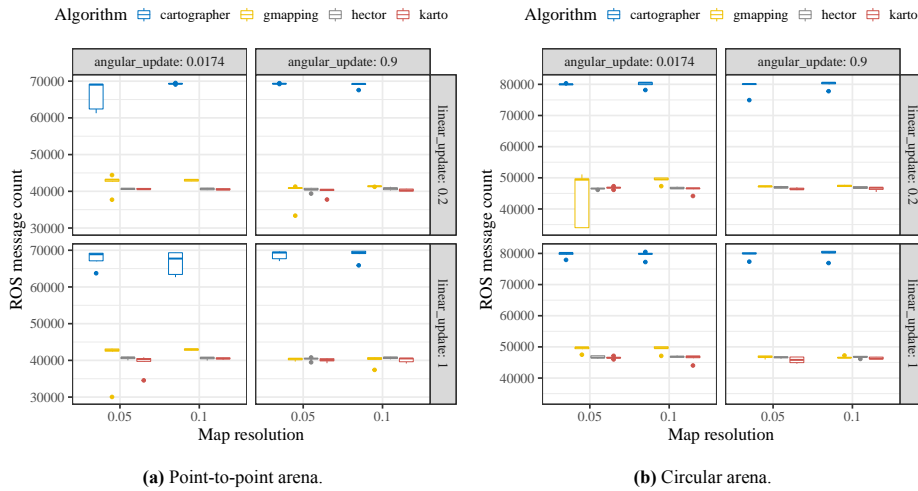


Figure 14. Message rate grouped by parameter values. We observe that the parameter configuration of the algorithms does not seem to have a major impact on ROS message count. The tight boxes and histograms appear largely clustered by algorithm. The message count for Cartographer is larger than for the other algorithms.

distribution. This is confirmed by a significant Shapiro-Wilk test of normality ($p = 3.109206 \cdot 10^{-19}$). Levene's test is significant as well ($p = 0.0005924397$), so its null hypothesis is rejected and homogeneity of variances may not be assumed. ANOVA assumptions are therefore violated. PERMANOVA results indicate a single significant main effect of the algorithm ($p < 2e^{-16}$).

In a Shapiro-Wilk test 3 out of 4 algorithms yield a significant p-value, substantiated by non-normal Q-Q plots. Thus the assumptions for Tukey's HSD test are violated. Dunn's test with Benjamini-Hochberg correction indicates a significant difference in message count between all pairs of algorithms except Hector-Karto.

Result: ROS message count - Point to point arena

From the four algorithms, three groups can be identified:

- Measurements for **Hector** and **Karto** are not significantly different. Their combined average message count is 46552.93;
- The average message count for **Gmapping** is 47430.38 (+1.9%);
- The average message count for **Cartographer** is 79666.23 (+71.1%).

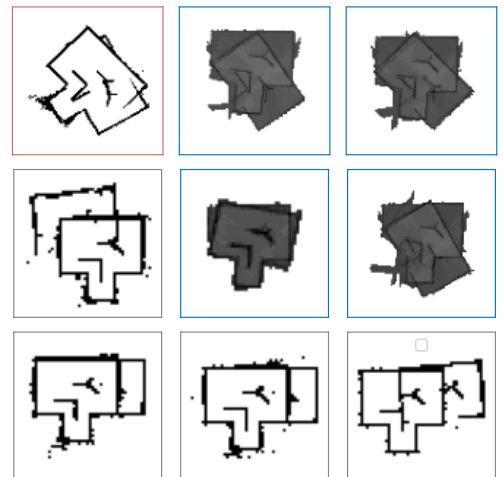


Figure 15. Erroneous maps over experiment executions on the circular arena. Since no error happened in the point-to-point arena, the algorithms seem to be experiencing difficulty with aligning the new scans in previously explored areas. Gmapping and Karto are the most robust algorithms, where Gmapping does not exhibit any error, while Karto presents only a minor one (figure marked in red).

7.5 Map Quality

This section presents the results related to **RQ2**: “What are the trade-offs between resource consumption and the accuracy of generated maps for SLAM algorithms in ROS-based systems?”

7.5.1 Visual inspection

The full collection of maps created in the experiment is available in the replication package. Some maps created by Cartographer are significantly darker than others, which is caused by the range of output values each cell in a Cartographer map can have. Whereas the other algorithms yield maps with cells that only have any of three possible values (i.e., unknown, occupied, or free) Cartographer outputs a *chance* of occupancy (in percent) for each cell. Further analysis showed that the majority of these occurrences emerge from runs where the angular update threshold factor was high. This resulted in fewer laser scans being processed and consequently less data to conclude from. Despite uncertainty about the area as a whole, these maps still show a clear distinction between the floor and walls of the arena (which are mostly assigned an occupancy probability approaching full certainty).

Considering the alignment of the walls within all maps, in Figure 15 there are multiple spurious walls or rotational alignment errors. These errors occur only in maps for the circular arena, giving the impression that the algorithms are experiencing difficulty with aligning the new scans in previously explored areas (i.e., when performing loop closure). Interestingly, none of the maps created by Gmapping show such errors. This gives reason to believe that Gmapping together with Karto (only a single, minor mapping error exhibited marked in red in Figure 15) are the most robust algorithms in the selection. The cartographer and Hector both created four severely mismatched maps. All errors for Cartographers are associated with a particular combination of high angular and high linear update thresholds.

By means of visual inspection, it seems that Karto creates the most consistent and high-fidelity maps among the four algorithms. Walls often appear thin and straight, whereas the maps created by other algorithms regularly contain spurious occupied cells and rounded walls. The most extreme example of this is Cartographer, which shows most protrusions outside of the arena itself. Maps created by Hector can be characterized by their eroded walls, with half-isolated occupied cells strongly contrasting their direct neighbors along the surface.

To support the visual analysis by quantitative means, the remainder of this section considers the quality of the created maps using the metrics framework by Filatov *et al.* [2017]. An assumption for using this framework is that all evaluated maps are visually consistent and cover the same area. Since only 9 out of 320 maps created in this experiment are strongly erroneous, this assumption holds.

Result: Map quality - Visual inspection

- **Gmapping** is the only algorithm for which no alignment errors occurred, making it the most robust algorithm in the evaluation (along with **Karto**, which created a *partially* skewed map in one single run);
- **Hector** and **Cartographer** both created four significantly misaligned maps;
- Maps created by **Cartographer** suffered the most from a high angular update threshold: the reduced processing frequency of incoming laser scans caused by this parameter resulted in large areas of uncertainty;
- Except the map resolution factor (which has a trivial effect on the created maps) variations in the algorithm configurations did not have prominently observable effects for the other algorithms;
- Overall, the cleanest maps by means of visual inspection are those created by **Karto** (featuring thin, straight walls with the least noise and artifacts).

7.5.2 Occupancy

The first metric considered is a map’s *occupancy* ratio. The rationale behind this metric is that accurate walls are thin and of clear contrast with unoccupied neighboring cells, as opposed to thick and blurry. This means that a lower occupancy ratio signifies higher map quality. To enable a fair comparison, the probabilistic maps created by the Cartographer are first binaries using the mean value of all cells as a threshold [Filatov *et al.*, 2017]. Subsequently, the ratio of occupied cells as part of the entire explored area is computed, as shown in Figure 16.

An outstanding observation is that a map resolution of 0.1 m/pixel results in approximately 50% higher occupancy ratio compared to a resolution of 0.05 m/pixel for most algorithms. Given that thicker walls are an inherent effect of a coarser granularity, this is a trivial effect. Gmapping, Hector and Karto show similar occupancy ratios, with no apparent influence of the configurations of the other parameters. The obtained measurements for Cartographer are consistently higher than for the other algorithms, then Cartographer maps are of inferior quality when it comes to the accuracy of walls. Furthermore, the erroneous Cartographer maps that were identified in Figure 15 show their impact on this metric, resulting in a large standard deviation in the lower right figure for the circular arena at a (high) map resolution of 0.05 m/pixel.

7.5.3 Corner count

Next, the *corner count* of each created map is considered. This is indicative of a map’s precision: lower quality maps have more corners than accurate ones (e.g., due to overlapping parts or inconsistent curvature of straight walls) [Filatov *et al.*, 2017]. The amount of cells belonging to corners for each map is computed by applying *Harris’ corner detector* [Harris *et al.*, 1988] to the binaries images from the previous step (considering all unexplored cells as occupied). Figure 17 depicts the results.

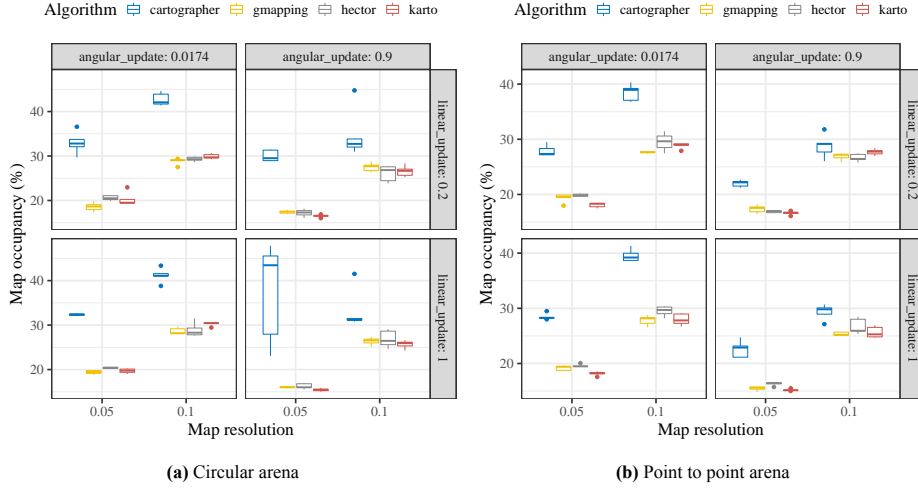


Figure 16. Occupancy ratio of the map where lower is better. Gmapping, Hector, and Karto display comparable occupancy ratios, with no significant influence from variations in other configuration parameters. In contrast, the occupancy measurements for Cartographer are consistently higher than those of the other algorithms, highlighting the inferior quality of its maps. The inaccuracies in the Cartographer maps, as depicted in Figure 15, contribute to this discrepancy, resulting in a large standard deviation in the final quarter of Figure 16a.

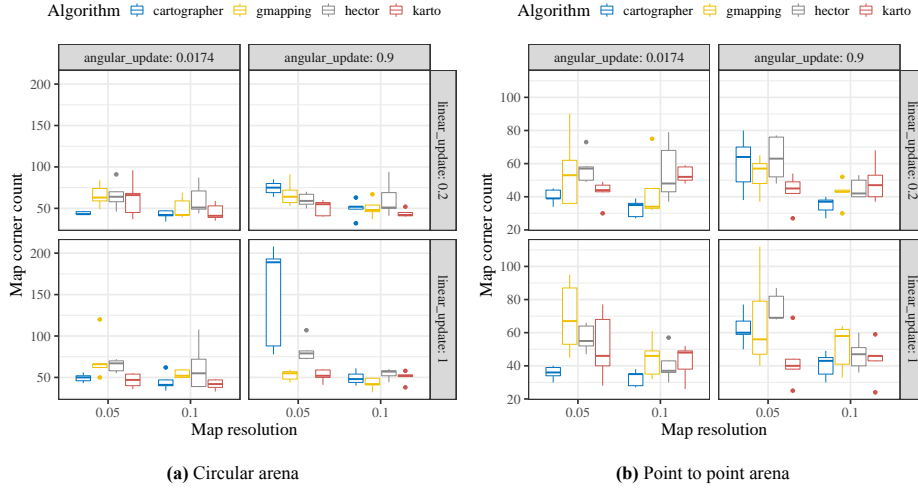


Figure 17. Corner count of the map where lower is better. The corner count is indicative of a map’s precision, where lower-quality maps have more corners than accurate ones. Hector, Gmapping, and Karto scored higher than Cartographer for the point-to-point arena. The arena was set up such that the laser sensor could never cast rays to any point outside of the enclosed walls; however, the algorithms wrongly interpreted some scans to have reached there.

An interesting observation regarding the corner count metric is that Hector, Gmapping, and Karto score higher than Cartographer for the point-to-point arena. The arena was set up such that the laser sensor could never cast rays to any point outside of the enclosed walls, however, the algorithms wrongly interpreted some scans to have reached there. As a result, spurious ‘explored’ cells *outside of the arena’s perimeter* are marked as unoccupied for these algorithms. Cartographers, being more reluctant to call free cells, did not suffer as much from this. As was the case for the previous metric, the erroneous maps for Cartographer as depicted in Figure 15 lead to diminishing scores in the circular arena (exhibiting a significantly higher corner count and standard deviation for the corresponding configuration).

7.5.4 Enclosed area count

A final quality metric is the *enclosed area count*. The idea here is the same as for the previous one: if there are overlapping rooms or artifacts in unexplored areas, the score will be

higher [Filatov *et al.*, 2017]. The amount of enclosed areas that could be detected using *Suzuki’s algorithm* [Suzuki *et al.*, 1985] for each configuration is depicted in Figure 18. To avoid polluting the data with minor disturbances that do not affect the usability of the map as a whole, only areas equivalent to 10x10cm or more are included.

Overall, the low map resolution of 0.1 m/pixel results in the fewest enclosed areas. This is a logical consequence of the fact that these maps are by definition more coarse. Variability among algorithms and across configurations is also low for this resolution. The high map resolution (0.05 m/pixel) does show significantly more enclosed areas for Gmapping and Hector. The reason for this is the same spurious rays as mentioned in Section 7.5.3.

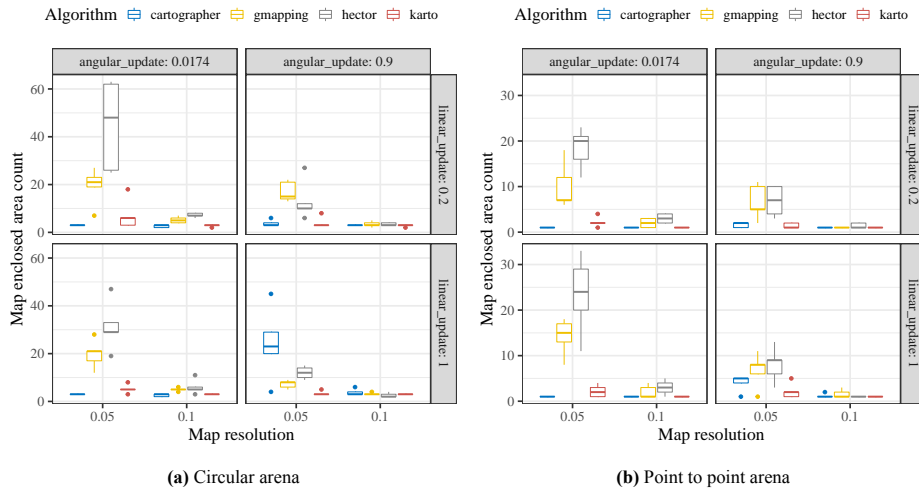


Figure 18. Enclosed area count of the map where lower is better. If there are overlapping rooms or artifacts in unexplored areas, the score will be higher, and therefore the map quality will be poorer. Different parameter configurations seem not to have a great impact on that measurement, except for Hector and Gmapping.

Result: Map quality - Quantitative metrics

The quantitative metrics for evaluating map quality as proposed by Filatov *et al.* [2017] [Filatov *et al.*, 2017] support the results from visual inspection. Additionally, they revealed that maps created by **Gmapping**, **Hector**, and **Karto** commonly contained 'unoccupied' cells outside of the arena perimeter. Since the experimental setup prevented any laser rays from being cast there, the unjustified assignment of these cells originates from spurious associations within the algorithm. Similar spurious alignments occurred for **Cartographer** as well, yet cells were assigned a very low probability of vacancy in such cases.

8 Discussion

Following the experimental results presented in Section 7, this section considers the insights from this study for both groups within this target audience.

8.1 Insights for Researchers

The results presented in Section 7.2 show that Hector imposed the smallest CPU utilization within the experimental context of this study. This is also reflected in Hector's energy consumption (Section 7.1). While these considerations advocate in favor of some advantageous aspects of Hector's design, Karto [Konolige *et al.*, 2010] yielded highly similar results. Statistical analysis could not indicate a significant difference between the two algorithms in terms of both, CPU utilization and energy consumption. This is interesting since Karto is designed according to a completely different SLAM paradigm based on pose graphs, while Hector can be classified as a Kalman filter/particle filter algorithm. The fact that the obtained measurements for these algorithms show no significant difference illustrates that the SLAM paradigm alone does not provide guarantees regarding an algorithm's

efficiency in terms of resources. Implementation-specific optimizations and design choices may significantly affect an algorithm's resource utilization as well.

Contrary to its advantageous CPU utilization and energy consumption, the average memory utilization of Hector was considerably higher than any of the other algorithms (up to 19.9% compared to Karto) and showed the highest relative increase over time (up to 6.8% throughout one run). This can be explained by Hector's approach to avoid getting stuck in local minima during gradient ascent: by using multiple maps, each at half the resolution of the preceding one and all simultaneously updated after the alignment process. Whereas other algorithms employ a similar technique, they often generate map levels from a single high-resolution map through down-sampling. To avoid this costly operation, Hector keeps different maps in memory. Gmapping Grisetti *et al.* [2007] is another example where decisions in the algorithm design have a significant impact on resource utilization. It is a particle filter approach and therefore moderately resource-intensive in theory, yet its implementation depends on the angular update threshold to such an extent that this parameter alone determined whether CPU utilization and energy consumption were among the best or worst-performing group of all algorithms. This can be explained by Gmapping's usage of FastSLAM Montemerlo *et al.* [2002]. Contrary to what the name suggests, the extensive reliance on Kalman filters makes this method slower than the other approaches. This can also be concluded from the increase in Gmapping in the CPU utilization over time. As a result, lowering the angular update threshold may trigger a new update immediately after the preceding one finishes. Keeping the CPU in this elevated state for a prolonged period consequently impacts energy consumption. Both, Hector's high memory utilization and the susceptibility of Gmapping's to a single parameter, are examples of design choices that have an immediate negative impact on resource utilization. It is important to consider such implications and the trade-offs they impose during algorithm design.

Graph-based approaches come with substantial theoretical advantages over more traditional methods (e.g., space-

Table 10. Relative *ranking* of subjects across evaluated metrics (lower is better).

Name	Energy consumption	CPU utilization	Memory utilization	ROS message count	Map quality
Cartographer	2	1	3	3	4
Gmapping	1 / 2 *	1 / 2 *	2	2	2
Hector	1	1	4	1	3
Karto	1	1	1	1	1

* = depends on parameter configuration

efficiency and widely studied optimization methods [Stachniss *et al.*, 2016]) making it a popular paradigm in current research on SLAM. However, in the context of this experiment, the two evaluated graph-based algorithms are positioned at opposite ends of the rankings for CPU utilization and energy consumption (as shown in Table 10). At almost 1.5 times the CPU utilization of Hector and Karto, Cartographer [Konolige *et al.*, 2010] is the most demanding algorithm in the evaluation. An explanation for this is that Cartographer is the most advanced of the algorithms in terms of e.g., sensor usage (including IMU data in sensor fusion) and map refinement (performing explicit loop closure and sub-map optimizations). Furthermore, it uses the ROS middleware for message exchange most extensively at over 65% more messages published to topics compared to the other algorithms. Again, these results show that the graph-based solution paradigm alone is no guarantee for resource efficiency. However, since Karto has proven to be the most efficient algorithm for every considered metric in this evaluation, *potential* resource utilization benefits of graph-based optimization techniques are apparent.

8.2 Insights for Practitioners

Practitioners working on ROS-based systems have a range of SLAM algorithms at their disposal. This section presents recommendations for choosing which algorithm to use, based on accuracy and resource utilization trade-offs and implementation requirements that should be considered. To gain an impression of how the evaluated algorithms compare to each other, their relative rank across all metrics is shown in Table 10.

8.2.1 Choosing a SLAM package

All four algorithms evaluated in this study were capable of creating a valid map of the environment in most cases. Visual inspection supported by a quantitative analysis concludes that Karto creates the most visually consistent maps, featuring thin, straight walls at accurate angles. While Cartographer and Hector both yielded the most erroneous maps, this does not invalidate their potential use in practice. Gmapping has proven to be the most robust algorithm in the context of this experiment, with no erroneous maps created.

Some algorithms employ sensor-fusing techniques to improve the accuracy of created maps and pose estimates. Cartographer, Gmapping, and Karto all use odometry data for this, which estimates the displacement of the robot based on the rotation of its wheels. Conversely, Hector relies solely on laser scan data. While this did not lead to lower resource utilization compared to the other algorithms, it can still be

a useful property if e.g., odometry data is unreliable. A potential threat to the scalability of this algorithm was found in the memory utilization of Hector: it was already higher than the other algorithms and showed the highest relative increase throughout the run.

In addition to odometry data, Cartographer uses the IMU sensor and advanced map refinement procedures aimed at improving map accuracy. However, the benefits of this did not show in the experimental context of this study. Possibly their effects are more apparent in diverse environments in the real world as compared to the controlled environment of the arenas used here. One advantage of Cartographer is that its created map contains occupancy probabilities for each cell, whereas the other algorithms create binary maps. Furthermore, it has proven to be more reluctant to call cells as unoccupied with limited data. These properties may make Cartographer suitable for occasions where the added granularity and reliability of the SLAM output outweigh the additional resource utilization: at an average CPU utilization of around 43% on the Raspberry Pi 4 fitted on the robot, it was the most computationally demanding of the four algorithms under evaluation.

8.2.2 Parameter tuning recommendations

All evaluated SLAM algorithms offer a wide range of parameters that can be tuned at run-time. The aim of these is to accommodate the algorithms to specific implementations by the user. Apart from configuring the parameters related to the laser sensor (which may differ based on the specific model used) the algorithms all worked 'out-of-the-box' without further parameter tuning. However, since tuning parameters may improve SLAM results in practice, the impact on resource utilization of four parameters that all algorithms had in common was evaluated in this study.

The *linear and angular update thresholds* define a minimum distance or rotational angle that the robot should travel before new laser scans are processed. While lowering these thresholds causes more frequent map updates and could thus theoretically be more demanding of the system's computational resources, no such effect was visible for most algorithms. Gmapping is the only algorithm whose energy consumption and CPU utilization were significantly affected by the angular update threshold: when this threshold is very low (0.0174rad in the case of this study) new updates are triggered almost immediately after the previous scans have been processed, since processing one set of scans takes longer for Gmapping than for the other algorithms. The prolonged computational load on the CPU also has an effect on the energy consumption of the system.

The *map resolution* was considered as an additional factor

in the experiment. All algorithms use a resolution of 0.05 m/pixel by default, but lowering this to 0.10 m/pixel was explored as a potential method to reduce resource utilization at the cost of accuracy. In practice, the obtained results showed no significant difference between the two resolutions. It is thus not recommended to reduce the resolution compared to the default.

Besides the parameters considered in this study, implementation-specific parameters are available for each algorithm. These can be used to further tailor the SLAM configuration. These include the number of particles in Gmapping or the number of sub-maps for Hector. These parameters were not evaluated in this study but may affect the resource utilization for the various algorithms.

8.2.3 Practical Implications

The findings from this study have important implications for deploying SLAM algorithms in real-world ROS-based robotic systems. Each algorithm demonstrates trade-offs that can guide users in selecting the most appropriate package for their needs. For instance, Karto's ability to generate accurate and visually consistent maps makes it a choice for environments that demand high precision. However, its dependency on odometry data means that environments with poor odometry (such as slippery or deformable terrains) may favor the choice for other algorithms that rely exclusively on laser scan data, such as Hector. In contrast, Gmapping stands out due to its robustness and lower error rates, which makes it a good choice when reliability is a concern, especially with minimal parameter tuning. Finally, The higher computational demands of the Cartographer, particularly for its map refinement techniques, suggest that it may still be a choice for systems with sufficient processing capacity, i.e., more powerful processors than the Raspberry Pi 4 used in this study, and with no concern on energy consumption.

8.3 SLAM Energy Footprint

Comparing the results for CPU utilization and energy consumption obtained in this experiment, they tend to correlate: higher CPU utilization tends to co-occur with higher energy consumption. Indeed, Spearman's correlation coefficient confirms a moderate positive correlation between the two metrics at $\rho = 0.6188474$ and $\rho = 0.6809241$ for the point-to-point and circular arena, respectively (note that the arenas directly affect energy consumption due to their difference in traversal time). In practice, the implications of the increased CPU utilization and energy consumption should be considered according to the specific implementation context. The difference in energy consumption between the various algorithms has a smaller order of magnitude than the difference in CPU utilization. A relative increase in CPU utilization of up to 47% resulted in no more than a 3% increase in energy consumed. Given these observations, other components in the robotic system presumably have a much larger impact on its energy consumption (e.g., the sensors and/or the WiFi adapter).

9 Threats To Validity

This section presents several threats to the validity of this study as well as measures taken to mitigate them. Threats are classified according to the framework proposed by Wohlin *et al.* [2012].

9.1 Internal Validity

The environment in which the experiment takes place may affect the investigated dependent variables. To mitigate these environmental effects, the experiment was performed in a controlled indoor environment. The arenas used did not change throughout the experiment, and the starting location for the robot was marked on the ground such that it would always be the same. Moreover, possible effects of environmental variability that may still arise are mitigated by executing all runs in random order.

Since energy is a dynamic resource and the performance of the robotic system may be affected by diminishing battery charge, the robot is equipped with a fully charged battery before each run. Furthermore, a cool-down period of one minute was inserted to stabilize the hardware conditions between runs.

The methods used for collecting the various metrics have been chosen according to established practices. CPU and memory measurements are obtained using PSUtil, which is the de-facto Python library for this task. ROS messages published on the system are logged with rosbag, which is the de-facto tool for this purpose. The INA219 sensor that was used to collect the power measurements for the robotic system has proven to be suitable for similar experimental setups in existing studies [Malavolta *et al.*, 2021; Peng *et al.*, 2019].

To ensure timestamps for the power measurements collected by the custom Arduino circuit are consistent with the other measurements collected on the robotic system's SBC (a Raspberry Pi) itself, the Arduino announces its local time in milliseconds as a response to the calling routine on the Raspberry Pi. The serial connection allows for transmission at a baud rate of 11500 bytes/second, thus the time taken for the response is negligible. The obtained local time is subsequently used to compute a time delta between the two platforms and synchronize the timestamps. While the preliminary test did show that the internal clock of the Arduino ticked slightly faster than that of the Raspberry Pi, the time drift on the Arduino was limited to 0.9 milliseconds per second (i.e., for every elapsed second on the Raspberry Pi, the Arduino's clock elapsed 1.0009 seconds). Given the short duration of the runs for this experiment, the effect of this time drift is considered negligible.

9.2 External Validity

This study considered only SLAM algorithms available as packages in ROS. Since ROS is just one out of many robotics middlewares in existence nowadays, there exist SLAM algorithms other than the ones evaluated here. The choice for this limitation to ROS can be justified by the fact that it has become the de-facto standard in robotics research and industry.

The results presented in this study are thus representative of most real application contexts.

Within the ROS package index, one additional package offering laser-based SLAM exists. SLAM Toolbox¹⁷ is a relatively new SLAM package, expanding on the Karto SLAM package. Since the latter was already included in the evaluation and installations of the two algorithms are incompatible (i.e., one cannot simultaneously have Karto and SLAM toolbox installed), SLAM Toolbox was not included in this study.

9.3 Construct Validity

While the mission is executed on the robotic system, it does not communicate with the remote PC orchestrating the experiment. Instead, it operates fully autonomously and re-initiates contact only upon termination of the mission. While the SSH connection between the two persists, any output that would normally be displayed on the remote terminal is instead logged to the disk, to avoid costly network traffic that could insert noise to the measurements.

The network to which the remote PC and robotic system are connected during the experiment is isolated from the internet and has no other clients connected to it. Furthermore, automatic updates and background services on the robotic system have been disabled where possible.

9.4 Conclusion Validity

Potential threats to the validity of conclusions presented in this study could be the unjustified usage of vehicles for statistical analysis. To ensure the validity of results, any assumptions such as normality of residuals or homogeneity of variance were explicitly checked before applying any statistical tests. If any of the assumptions were violated (even after transforming the data), the analysis resorted to non-parametric alternatives.

A potential threat to the validity of our results arises from the lack of normality in the data. To address this, we employed PERMANOVA, a non-parametric alternative that does not rely on normality assumptions, thereby reducing the risk of drawing invalid conclusions from non-normal data. Furthermore, we rigorously tested for homogeneity of variances using Levene's test, which confirmed that the assumption of equal variances was upheld ($p = 0.545$). By using 100 million random permutations in the PERMANOVA analysis, we minimized the influence of random variation, ensuring the robustness of our results. Post-hoc analysis was conducted using Dunn's test with Benjamini-Hochberg correction, which similarly does not require normality, further preventing misjudgment. Despite these measures, we acknowledge that caution should be exercised when interpreting the results, particularly in instances where transformations failed to normalize the data.

10 Conclusion

SLAM is essential in the context of robotic systems and has been widely studied in research over the last three decades. Despite SLAM can be implemented by several paradigms, in this study, we only consider popular 2D laser-based implementations that are available in the Robot Operating System (ROS).

Our study is based on two main research questions: **RQ1**: *How do SLAM algorithms impact resource utilization of ROS-based systems?*; and **RQ2**: *What are the trade-offs between resource consumption and the accuracy of generated maps for SLAM algorithms in ROS-based systems?*.

While investigating *RQ1*, the results showed that implementation-specific SLAM design choices are decisive in the resource utilization of an algorithm.

A concrete example is the distinctive manner of *Cartographer* relying on *sensor fusion* and advanced map refinement techniques, which makes it resource greedy ($\approx 65\%$ more than the most CPU-efficient). Through the wide range of other measurements and statistical analysis, this study aids *researchers* to gain an understanding of how SLAM algorithms from various paradigms impact the resource utilization of ROS-based systems.

RQ2 is of special interest for *practitioners*, as it highlights an important consideration when choosing which of the SLAM algorithms available in the ROS ecosystem to use for their application. In addition to the aforementioned metrics, the results regarding map quality are relevant to this question. In the experimental context of this study, *Karto* created the most accurate maps (after visual inspection and with fewer errors) and imposed the lowest CPU usage of the system ($\approx 40\%$ less than the less CPU-efficient). Therefore, we could not identify a consistent trade-off between resource utilization and map quality, requiring the practitioners to be careful depending on the specific implementation context. Further considerations presented in Section 8 can aid practitioners in making resource-aware design choices.

While this study focused exclusively on SLAM algorithms that rely on laser scan data, the mapping can rely on different sensors, and some of them are capable of performing SLAM in 3D. Therefore, as future work, potential research opportunities include an evaluation of SLAM performed using these various sensors impacts resource utilization. This study could also be extended to evaluating SLAM in a wide range of scenarios outside of a controlled lab environment and considering ROS 2.

Declaration

Acknowledgements

Funding

This research was partially funded by the Rijksdienst voor Ondernemend Nederland (RVO) through the ITEA3 BUMBLE project (18006), and the CNPQ/FA through the PPP-CP-20/2018 call.

¹⁷http://wiki.ros.org/slam_toolbox

Authors' Contributions

Engel Hamer: Investigation, Experimental Implementation, Data Curation, Writing – original draft. Michel Albonico: Data curation, Visualization, Writing – review & editing. Ivano Malavolta: Conceptualization, Methodology, Investigation, Writing – review editing, Supervision.

Competing interests

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Michel Albonico reports a relationship with Federal Technological University of Paraná - Campus Francisco Beltrão that includes: employment. Ivano Malavolta reports a relationship with Vrije Universiteit Amsterdam that includes: employment.

Availability of data and materials

The datasets generated and analyzed during the current study are available in the following GitHub repository: <https://github.com/IntelAgir-Research-Group/thesis-replication-package>

References

- Abdelrasoul, Y., Saman, A. B. S. H., and Sebastian, P. (2016). A quantitative study of tuning ros gmapping parameters and their effect on performing indoor 2d slam. In *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 1–6. DOI: 10.1109/ROMA.2016.7847825.
- Abdi, H. and Williams, L. J. (2010). Tukey's honestly significant difference (hsd) test. *Encyclopedia of research design*, 3(1):1–5. Available at: <https://personal.utdallas.edu/~Herve/abdi-HSD2010-pretty.pdf>.
- Aerts, P. and Demeester, E. (2017). Benchmarking of 2d-slam algorithms. In *Ad Usus Navigantium*. Available at: http://www.acro.be/downloadvrij/Benchmark_2D_SLAM.pdf.
- Albonico, M., Malavolta, I., Pinto, G., Guzman, E., Chinnappan, K., and Lago, P. (2021). Mining energy-related practices in robotics software. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 483–494. DOI: 10.1109/MSR52588.2021.00060.
- Alcalde, M., Ferreira, M., González, P., Andrade, F., and Tejera, G. (2022). Da-slam: Deep active slam based on deep reinforcement learning. In *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, pages 282–287. DOI: 10.1109/LARS/SBR/WRE56824.2022.9996006.
- Alharbi, M. and Alshayeb, M. (2024). An empirical investigation of the relationship between pattern grime and code smells. *Journal of Software: Evolution and Process*, page e2666. DOI: 10.1002/smr.2666.
- Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. *Austral ecology*, 26(1):32–46. DOI: 10.1111/j.1442-9993.2001.01070.pp.x.
- Andert, F. and Mosebach, H. (2019). Lidar slam positioning quality evaluation in urban road traffic. In *International Conference on Intelligent Transport Systems*, pages 277–291. Springer. DOI: 10.1007/978-3-030-38822-5_19.
- Balaguer, B., Balakirsky, S., Carpin, S., and Visser, A. (2009). Evaluating maps produced by urban search and rescue robots: lessons learned from robocup. *Autonomous Robots*, 27(4):449–464. DOI: 10.1007/s10514-009-9141-z.
- Basili, V., Caldiera, G., and Rombach, H. (1994). Goal question metric paradigm. *Encyclopedia of software engineering*, 1:528–532. Available at: <https://www.cs.umd.edu/users/basili/publications/technical/T87.pdf>.
- Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300. DOI: 10.1111/j.2517-6161.1995.tb02031.x.
- Brown, M. B. and Forsythe, A. B. (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346):364–367. DOI: 10.1080/01621459.1974.10482955.
- Cabane, H. and Farias, K. (2024). On the impact of event-driven architecture on performance: An exploratory study. *Future Generation Computer Systems*, 153:52–69. DOI: 10.1016/j.future.2023.10.021.
- Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. (2016). Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332. DOI: 10.1109/TRO.2016.2624754.
- Castaño, J., Martínez-Fernández, S., Franch, X., and Bogner, J. (2023). Exploring the carbon footprint of hugging face's ml models: A repository mining study. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12. DOI: 10.1109/ESEM56168.2023.10304801.
- Cheeseman, P., Smith, R., and Self, M. (1987). A stochastic map for uncertain spatial relationships. In *4th international symposium on robotic research*, pages 467–474. MIT Press Cambridge. Available at: https://www.researchgate.net/publication/234808346_A_stochastic_map_for_uncertain_spatial_relationships.
- Chong, T., Tang, X., Leng, C., Yogeswaran, M., Ng, O., and Chong, Y. (2015). Sensor technologies and simultaneous localization and mapping (slam). *Procedia Computer Science*, 76:174–179. DOI: 10.1016/j.procs.2015.12.336.
- Costa, L. R. and Colombini, E. L. (2023). Can semantic-based filtering of dynamic objects improve visual slam and visual odometry? In *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*, pages 567–572. DOI: 10.1109/LARS/SBR/WRE59448.2023.10332956.
- De Mello Gai, A., Bevilacqua, S., Cukla, A. R., and Gamarra, D. F. T. (2023). Evaluation on imu and odometry sensor fu-

- sion for a turtlebot robot using amcl on ros framework. In *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*, pages 637–642. DOI: 10.1109/LARS/SBR/WRE59448.2023.10332977.
- Dhaoui, R. and Rahmouni, A. (2022). Mobile robot navigation in indoor environments: Comparison of lidar-based 2d slam algorithms. In *Design Tools and Methods in Industrial Engineering II: Proceedings of the Second International Conference on Design Tools and Methods in Industrial Engineering, ADM 2021, September 9–10, 2021, Rome, Italy*, pages 569–580. Springer. DOI: 10.1007/978-3-030-91234-5_7.
- Dordevic, M., Hamer, E., and Malavolta, I. (2021). Software engineering research on the robot operating system: A systematic mapping study. DOI: 10.1016/j.jss.2022.111574.
- Duchoň, F., Hažík, J., Rodina, J., Tölgyessy, M., Dekan, M., and Sojka, A. (2019). Verification of slam methods implemented in ros. *Journal of Multidisciplinary Engineering Science and Technology (JMEST)*. Available at: <https://www.jmest.org/wp-content/uploads/JMESTN42353033.pdf>.
- Dunn, O. J. (1964). Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252. DOI: 10.2307/1266041.
- Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110. DOI: 10.1109/MRA.2006.1638022.
- Estefo, P., Simmonds, J., Robbes, R., and Fabry, J. (2019). The robot operating system: Package reuse and community dynamics. *Journal of Systems and Software*, 151:226–242. DOI: 10.1016/j.jss.2019.02.024.
- Filatov, A., Filatov, A., Krinkin, K., Chen, B., and Molodan, D. (2017). 2d slam quality evaluation methods. In *2017 21st Conference of Open Innovations Association (FRUCT)*, pages 120–126. IEEE. DOI: 10.23919/FRUCT.2017.8250173.
- Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, 23(1):34–46. DOI: 10.1109/TRO.2006.889486.
- Harris, C., Stephens, M., et al. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Cite-seer. Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=88cdfbeb78058e0eb2613e79d1818c567f0920e2>.
- Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE. DOI: 10.1109/ICRA.2016.7487258.
- Kleiner, A. and Dornhege, C. (2007). Real-time localization and elevation mapping within urban search and rescue scenarios. *Journal of Field Robotics*, 24(8-9):723–745. DOI: 10.1002/rob.20208.
- Kohlbrecher, S., Meyer, J., von Stryk, O., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. DOI: 10.1109/SSRR.2011.6106777.
- Kolak, S., Afzal, A., Le Goues, C., Hilton, M., and Timperley, C. S. (2020). It takes a village to build a robot: An empirical study of the ros ecosystem. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 430–440. DOI: 10.1109/IC-SME46990.2020.00048.
- Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., and Vincent, R. (2010). Efficient sparse pose adjustment for 2d mapping. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 22–29. IEEE. DOI: 10.1109/IROS.2010.5649043.
- Koziolek, H., Grüner, S., and Rückert, J. (2020). A comparison of mqtt brokers for distributed iot edge computing. In Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., and Zimmermann, O., editors, *Software Architecture*, pages 352–368, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-58923-3_23.
- Le, X. S., Fabresse, L., Bouraqadi, N., and Lozenguez, G. (2018). Evaluation of out-of-the-box ros 2d slams for autonomous exploration of unknown indoor environments. In Chen, Z., Mendes, A., Yan, Y., and Chen, S., editors, *Intelligent Robotics and Applications*, pages 283–296, Cham. Springer International Publishing. DOI: 10.1007/978-3-319-97589-4_24.
- Lee, S. U., Fernando, N., Lee, K., and Schneider, J.-G. (2024). A survey of energy concerns for software engineering. *Journal of Systems and Software*, 210:111944. DOI: 10.1016/j.jss.2023.111944.
- Lins, F. C. A., Luz, O. M., Medeiros, A. A. D., and Alsina, P. J. (2020). A comparison of three different parameterizations to represent planes for the mapping step of direct visual slam approaches. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, pages 1–6. DOI: 10.1109/LARS/SBR/WRE51543.2020.9306932.
- Malavolta, I., Chinnappan, K., Swanborn, S., Lewis, G. A., and Lago, P. (2021). Mining the ros ecosystem for green architectural tactics in robotics and an empirical evaluation. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 300–311. IEEE. DOI: 10.1109/MSR52588.2021.00042.
- Malavolta, I., Lewis, G., Schmerl, B., Lago, P., and Garland, D. (2020). How do you architect your robots? state of the practice and guidelines for ros-based systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 31–40. IEEE. DOI: 10.1145/3377813.3381358.
- Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598. Available at: <https://cdn.aaai.org/AAAI/2002/AAAI02-089.pdf>.
- Nardi, L., Bodin, B., Zia, M. Z., Mawer, J., Nisbet, A., Kelly, P. H. J., Davison, A. J., Luján, M., O’Boyle, M. F. P., Riley, G., Topham, N., and Furber, S. (2015). Introduc-

- ing slambench, a performance and accuracy benchmarking methodology for slam. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5783–5790. DOI: 10.1109/ICRA.2015.7140009.
- Ngoc, H. T., Vinh, N. N., Nguyen, N. T., and Quach, L.-D. (2023). Efficient evaluation of slam methods and integration of human detection with yolo based on multiple optimization in ros2. *International Journal of Advanced Computer Science & Applications*, 14(11). Available at: <https://pdfs.semanticscholar.org/5168/13c9d909c73576bf6d4d6c923095f3226a2f.pdf>.
- Nguyen, Q. H., Johnson, P., and Latham, D. (2022). Performance evaluation of ros-based slam algorithms for handheld indoor mapping and tracking systems. *IEEE Sensors Journal*, 23(1):706–714. DOI: 10.1109/JSEN.2022.3224224.
- Đorđević, M., Albonico, M., Lewis, G. A., Malavolta, I., and Lago, P. (2023). Computation offloading for ground robotic systems communicating over wifi—an empirical exploration on performance and energy trade-offs. *Empirical Software Engineering*, 28(6):140. DOI: 10.1007/s10664-023-10351-6.
- Partap, A., Grayson, S., Huzaifa, M., Adve, S., Godfrey, B., Gupta, S., Hauser, K., and Mittal, R. (2022). On-device cpu scheduling for robot systems. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11296–11303. DOI: 10.1109/IROS47612.2022.9982085.
- Peng, T., Zhang, D., Liu, R., Asari, V. K., and Loomis, J. S. (2019). Evaluating the power efficiency of visual slam on embedded gpu systems. In *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, pages 117–121. DOI: 10.1109/NAECON46414.2019.9058059.
- Quigley, M., Faust, J., Foote, T., Leibs, J., et al. (2009). Ros: an open-source robot operating system. Available at: <https://ai.stanford.edu/~mquigley/papers/icra2009-ros.pdf>.
- Roesler, O. and Ravindranath, V. P. (2020). Evaluation of slam algorithms for highly dynamic environments. In Silva, M. F., Luís Lima, J., Reis, L. P., Sanfeliu, A., and Tardioli, D., editors, *Robot 2019: Fourth Iberian Robotics Conference*, pages 28–36, Cham. Springer International Publishing. DOI: 10.1007/978-3-030-36150-1_3.
- Rojas-Fernández, M., Mújica-Vargas, D., Matuz-Cruz, M., and López-Borreguero, D. (2018). Performance comparison of 2d slam techniques available in ros using a differential drive robot. In *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pages 50–58. DOI: 10.1109/CONIELECOMP.2018.8327175.
- Samarakoon, K. Y., Pereira, G. A. S., and Gross, J. N. (2022). Impact of the trajectory on the performance of rgb-d slam executed by a uav in a subterranean environment. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 812–820. DOI: 10.1109/ICUAS54217.2022.9836199.
- Sankalprajan, P., Sharma, T., Perur, H. D., and Sekhar Pagala, P. (2020). Comparative analysis of ros based 2d and 3d slam algorithms for autonomous ground vehicles. In *2020 International Conference for Emerging Technology (INCET)*, pages 1–6. DOI: 10.1109/INCET49848.2020.9154101.
- Santos, J. M., Portugal, D., and Rocha, R. P. (2013). An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6. IEEE. DOI: 10.1109/SSRR.2013.6719348.
- Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611. DOI: 10.1093/biomet/52.3-4.591.
- Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68. DOI: 10.1177/027836498600500404.
- Snedecor, G. W., Cochran, W. G., et al. (1968). *Statistical methods*. Number 6th ed.(repr.). Iowa state university press. Book.
- Stachniss, C., Leonard, J. J., and Thrun, S. (2016). Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer. DOI: 10.1007/978-3-319-32552-1_46.
- Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. DOI: 10.1109/IROS.2012.6385773.
- Su, Z., Zhou, J., Dai, J., and Zhu, Y. (2020). Optimization design and experimental study of gmapping algorithm. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 4894–4898. DOI: 10.1109/C-CDC49329.2020.9164603.
- Suger, B., Tipaldi, G. D., Spinello, L., and Burgard, W. (2014). An approach to solving large-scale slam problems with a small memory footprint. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3632–3637. DOI: 10.1109/ICRA.2014.6907384.
- Suzuki, S. et al. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46. DOI: 10.1016/0734-189X(85)90016-7.
- Swanborn, S. (2020). An empirical evaluation of energy-efficient architectural tactics in robotics software.
- Swanborn, S. and Malavolta, I. (2021). Robot Runner: A Tool for Automatically Executing Experiments on Robotics Software. In *Proceedings of the ACM/IEEE 43rd International Conference on Software Engineering*. ACM. DOI: 10.1109/ICSE-Companion52605.2021.00029.
- Thrun, S. et al. (2002). Robotic mapping: A survey. Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=cbe046f24f31aace8b61b36e41392a93225029e0>.
- Vaussard, F., Rétornaz, P., Hamel, D., and Mondada, F. (2012). Cutting down the energy consumed by domestic robots: Insights from robotic vacuum cleaners. In *Advances in Autonomous Robotics*, pages 128–139, Berlin, Heidelberg. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-32527-4_12.
- Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B.,

- and Wesslen, A. (2012). Experimentation in software engineering - an introduction. DOI: 10.1007/978-3-662-69306-3.
- Yagfarov, R., Ivanou, M., and Afanasyev, I. (2018). Map comparison of lidar-based 2d slam algorithms using precise ground truth. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1979–1983. DOI: 10.1109/ICARCV.2018.8581131.
- Zhao, Y.-L., Hong, Y.-T., and Huang, H.-P. (2024). Comprehensive performance evaluation between visual slam and lidar slam for mobile robots: Theories and experiments. *Applied Sciences*, 14(9):3945. DOI: 10.3390/app14093945.
- Zia, M. Z., Nardi, L., Jack, A., Vespa, E., Bodin, B., Kelly, P. H., and Davison, A. J. (2016). Comparative design space exploration of dense and semi-dense slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1292–1299. DOI: 10.1109/ICRA.2016.7487261.