Mining the ROS ecosystem for Green Architectural Tactics in Robotics and an Empirical Evaluation

Ivano Malavolta[†], Katerina Chinnappan[†], Stan Swanborn[†], Grace A. Lewis[◇], Patricia Lago[†]
† Vrije Universiteit Amsterdam, The Netherlands – {i.malavolta | p.lago | k.p.chinnappan | s.o.swanborn}@vu.nl
◇ Software Engineering Institute, Carnegie Mellon University, USA – glewis@sei.cmu.edu

***Abstract*—In today's world, reducing energy consumption should be the goal for any organization and any system, including robotics software systems. However, state of the practice in robotics software development focuses primarily on achieving functionality and performance, with minimal recognition of energy-efficiency as a driving software quality. The goal of this paper is to identify, synthesize, and empirically evaluate architectural tactics for energy-efficiency applied by practitioners in real robotics projects. Four tactics were identified by mining software repository techniques applied to the ROS ecosystem. The tactics were evaluated via experimentation on a real, commodity robotics system. Results show that the application of green architectural tactics tends to largely improve the energy-efficiency of the robot (7.9% energy savings when all tactics are applied) and that the movement strategy and the physical environment where the robot operates strongly influence how energy is consumed by the robot.**

I. Introduction

As of today, energy is a critical resource for any software system, *e.g.,* in data centers [22] and mobile apps [53]. According to the International Technology Roadmap for Semiconductors [7], at the current rate of growth, the total energy required by ICT alone would be over 10^{20} Joules by 2040, *i.e.,* equivalent to today's global energy production.

Robotics systems play a non-negligible role in this growth [55]. For example, industrial robots consume about 8% of the total electricity in automotive assembly lines in the US [51]. Improving the energy-efficiency of robotics systems can lead to (i) substantial benefits in terms of environmental sustainability and (ii) an improvement of the quality of service of battery-operated robotics systems such as autonomous cars and factories, drones, service robots, etc.

In this context, it is important to understand that *software* is becoming the core aspect in robotics. For example, each new NASA exploration mission uses more control software than all its prior missions combined, with the Curiosity Mars Rover having 3M+ LOCs [39]. Despite the initiatives related to software engineering for robotics of the past years (*e.g.,* the IEEE TCSE for Robotics [10]), fundamental challenges are still affecting robotics software: (i) it is becoming more and more *large, complex, and difficult to write and debug* [31][32][73]; (ii) the development of robotics software is still a *craftsmanship* activity instead of following established engineering practices [35][48]; and (iii) the predominant focus on performance and functional aspects leads roboticists to neglect crucial software quality attributes [23][25], with *energy-efficiency glaringly missing* [14].

Our **goal** is to identify and empirically evaluate architectural tactics for energy-efficient robotics software emerging from the state-of-the-practice, or *green tactics* for short. Architectural tactics are *design decisions that influence the achievement of system qualities and can be reused across projects* [19]; *e.g.,* a tactic for energy-efficiency is to adapt the rate for sampling a sensor based on the energy-level of the robot. Today there is no solid body of knowledge on green tactics for robotics software, leaving roboticists far behind the state of the art in software development.

To fill this gap, we apply a mixed-method empirical research design composed of two main parts. First, we apply *software repository mining techniques* to extract a set of green tactics for robotics software. In this study we refer to robotics software as any software running on top of the Robot Operating System (ROS) [60]. ROS is the de-facto standard for robotics software [48] and it is used for aerial robots (*e.g.,* drones), ground robots (*e.g.,* autonomous cars and warehouse automation), industrial robots (*e.g.,* manipulator arms), etc. We use the ROS ecosystem to extract green tactics from real projects developed in real development contexts. Second, we perform an *empirical evaluation of the identified four green tactics*. Specifically, we implement each green tactic in a real robot running a common ROS software stack. Then, we carry out a quantitative assessment of the run-time impact of each tactic in terms of the energy consumption of the robot through different missions and physical environments. After performing 300 individual runs for a total of 10+ hours of sheer execution time, we provide evidence that (i) three out of four tactics lead to large energy savings, (ii) applying all tactics in combination leads to the best results in terms of energy consumption, and (iii) the movement strategy of the robot influences how energy is consumed.

The **main contributions** of this paper are: (i) a reusable dataset with several repositories, metadata, and discussion posts about ROS-robotics software, (ii) the identification of four green tactics for robotics software used in real projects, (iii) an empirical evaluation of the green tactics on a real robot performing different missions and in different physical environments, and (iv) the full replication package for the study. To the best of our knowledge, *this study is the first empirical investigation on green tactics for robotics software*.

Our **target audience** includes *roboticists* who want to use empirically-proven green tactics for architecting their next ROS-based system, and *researchers* who can use the green tactics as a foundation for defining new approaches to automatically improve the energy-efficiency of ROS-based systems. By providing quantitative evidence about the actual impact of the green tactics, we advocate more attention to energy-related aspects of robotics software. At the time of writing, this line of research has not yet been explored.

II. Robotics Software

ROS is the de-facto standard and key technological enabler for robotics software. It supports more than 140 types of robots and has a vibrant open-source ecosystem with many GitHub repositories containing ROS-based software, 4,152 publicly-available ROS packages, 7,696 ROS Wiki users, and 36,229 ROS Answers users [8][31].

The ROS ecosystem allows for the development of hardware components and robotics systems independently from their deployment environment (*e.g.,* Operating System (OS), programming language, hardware). As long as the development environment implements a mutually supported and compatible ROS version, the two components, whether only a hardware component added to a robotics system or a complete robotics system added to a cell of robotics systems, will be able to communicate with each other.

In ROS, **topics** allow for many-to-many writing or reading asynchronous, continuous data streams by *publishing* or *subscribing* to a topic, respectively; **actions** allow for one-to-one, synchronous and asynchronous client-to-server calls specifically meant for use with long lasting tasks; and **services** allow for one-to-one, client-to-server synchronous RPCs. These methods are the only communication methods available to ROS nodes, which significantly simplifies communication across nodes and allows one node to make safe assumptions about another node's access points.

III. Study design

To foster independent verification and replication, a full replication package of this study is publicly available [15].

A. Goal and Research Question

Following the template by Basili *et al.* [18], our **goal** is to analyse *the ROS ecosystem* for the purpose of *identifying and evaluating* a set of *architectural tactics* with respect to their *energy-efficiency* from the point of view of *roboticists and researchers* in the context of *open-source ROS-based systems*.
RQ1 – Which architectural tactics are applied in the development of energy-efficient robotics software? This research question is answered *qualitatively*; we identify, extract, and establish a set of four green tactics used in real-world robotics projects. The four tactics are synthesized via a multi-stage experimental study targeting multiple open-source robotics projects and their related artifacts. We expect that additional green tactics will emerge in the future (*e.g.,* by surveying roboticists or via systematic literature studies similar to [67]), using our results as a foundation based on the state-of-the-practice. The identified green tactics provide: (i) a unified vocabulary that roboticists and researchers can use for discussing energy-related design decisions; (ii) a catalogue of ready-to-use and validated solutions for energy-related design problems for future robotic projects; (iii) a foundation for future fundamental and applied research in energy-efficiency for robotics software (*e.g.,* finding techniques for automatically applying green tactics to ROS systems).
RQ2 – To what extent does the application of green tactics impact robotics software' energy-efficiency? This research question is answered *quantitatively*; for each green tactic we carry out an empirical assessment of its run-time impact in terms of energy consumption of a real robot. By answering this RQ, roboticists are offered objective data on how different green tactics can make their robotics systems more efficient.

B. Identifying the green tactics (RQ1)

Green tactics identification entails (see Table I and Fig. 1):
Phase 1 – in this phase we build a dataset containing as much ROS-related data as possible. Specifically, we mine the following data sources:

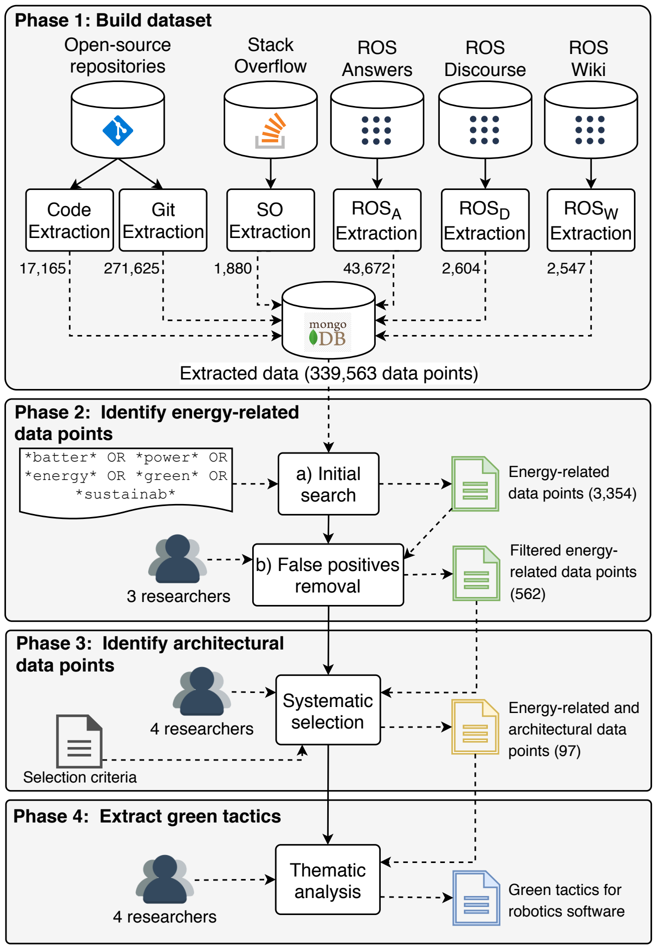
- *Open-source repositories*: We start from a publicly-available dataset of 335 Git repositories containing real open-source ROS systems [48]. We then (i) clone the repositories and extract *all* source code comments and Markdown files and (ii) crawl all pull requests/issues (incl. their discussions) and commit messages.
- *Stack Overflow*: We crawl all questions with the ROS tag, their answers, comments, and related metadata.
- *ROS Answers*: We crawl the same information as in SO for *all* its posts, answers, comments, and related metadata.
- *ROS Discourse*: We crawl all its posts, discussions, and related metadata.
- *ROS Wiki*: We crawl all its pages and related metadata.

In this study a data point can be a discussion on Stack Overflow, a GitHub pull request, a commit message, a source code file, etc. Given the heterogeneity of the extracted data, we develop a dedicated extractor for each type of data point. All extractors persist the extracted data into a single MongoDB database. We use MongoDB because (i) it stores data as *key-value pairs*, making it highly efficient and easy to inspect, and (ii) it is *schemaless*, allowing us to easily persist (and query) the mined data in a single database.

TABLE I: Summary of extracted data for answering RQ1

	Phase 1	**Phase 2 (a)**	**Phase 2 (b)**	**Phase 3**
Source code comments	16,069	172	55	14
Markdown files	1,096	86	21	4
Commit messages	218,385	665	209	38
Issues and PRs	53,310	915	69	25
SO discussions	1,880	32	3	0
ROS Answers discussions	43,672	1227	170	7
ROS Discourse discussions	2,604	197	12	7
ROS Wiki pages	2,547	60	23	2
Total	**339,563**	**3,354**	**562**	**97**

Phase 2 – In this phase we identify the data points mentioning energy-related topics. First (Phase 2a), we query the MongoDB database with the pattern expression reported in Figure 1 (we consider all possible combinations of both lower and upper case for each term in the pattern). We decided to use a



Phase 1: Build dataset
Open-source repositories
Stack Overflow
ROS Answers
ROS Discourse
ROS Wiki
Code Extraction
Git Extraction
SO Extraction
ROS$_A$ Extraction
ROS$_D$ Extraction
ROS$_W$ Extraction
17,165
271,625
1,880
43,672
2,604
2,547
mongoDB
Extracted data (339,563 data points)
Phase 2: Identify energy-related data points
batter OR *power* OR *energy* OR *green* OR *sustainab*
a) Initial search
Energy-related data points (3,354)
3 researchers
b) False positives removal
Filtered energy-related data points (562)
Phase 3: Identify architectural data points
4 researchers
Systematic selection
Energy-related and architectural data points (97)
Selection criteria
Phase 4: Extract green tactics
4 researchers
Thematic analysis
Green tactics for robotics software

Fig. 1: Phases for identifying the green tactics

pattern matching approach following the intuition that developers often use semantically similar or related keywords when considering energy aspects of their systems. The keyword-based strategy has proven to be a powerful solution for a number of mining problems [20] and has been successfully applied in previous studies on mining software repositories and Q&A platforms about energy-efficiency [24][29][49][52][58]. The keywords were identified by considering, analyzing, and combining mining strategies in previous empirical studies on software energy-efficiency [24][49][52][58].

Second (Phase 2b), we manually validate all 3,354 data points to reduce the number of false positives, *e.g.,* due to sentences such as "Problems with powering on a robot", which are clearly out of scope. The manual validation is performed in three rounds. In the first round we performed a stratified random selection of the data points (50 data points for each type of extracted data, *i.e.,* 50 commit messages, 50 GitHub issues, etc.) and two researchers independently classified them as either true or false positives, with a third researcher acting as arbiter in case of conflicts. The second round follows the same procedure, but is performed on another subset of the data points. During both rounds, disagreements were discussed to ensure high quality of the selected data points. Also, before discussing emerged conflicts, the level of agreement between the researchers is statistically assessed using Cohen's kappa coefficient [28], leading to nearly perfect levels of agreement (0.9 for round 1 and 0.8 for round 2), making us reasonably confident about the objectivity of our classification. Finally, based on the high level of agreement in the first two rounds, the last round involves a single researcher classifying all the remaining data points. The result of this filtering step is a set of 562 data points (fourth column of Table I).

Phase 3 – In this phase we perform an in-depth assessment of the 562 selected data points to consider only those discussing *architecturally-relevant concerns*. In this context, we use the definition of system concern from [41], where it is defined as the interest in a system relevant to one or more of its stakeholders (*e.g.,* presence of integrator nodes, system layers, interfaces to other systems). Inspired by the systematic literature review method [74], we *manually analyze* each data point and select those fulfilling a set of well-defined inclusion and exclusion criteria. Two examples of representative inclusion criteria are: (i) data points concerning a ROS architectural entity (*e.g.,* ROS nodes, topics, services), (ii) data points mentioning architecturally-relevant design decisions. The full set is available in the replication package. Three researchers are involved in this step and emerging conflicts are resolved by a fourth researcher. As in Phase 2, we perform this selection in three rounds and statistically assess the level of agreement via the Cohen Kappa statistics in rounds 1 and 2 (before discussing conflicts), with values of 1 and 0.925, respectively. The result is a set of 97 data points (last column of Table I).

Phase 4 – The 97 architecturally-relevant data points are analyzed to identify and extract green tactics. This phase is conducted by applying the *thematic analysis* methodology [30]. We chose thematic analysis because architectural information can be strongly dependent on project- and system-specific characteristics and thematic analysis copes well with context-dependent data [30][48][70]. Four researchers are involved in this phase, whose activities can be decomposed into four main sequential steps: (i) for each data point two researchers independently collect the list of mentioned architectural entities (*e.g.,* ROS nodes and topics) and extract a brief summary of the main design decisions related to energy consumption; (ii) three researchers independently analyze each data point in its context (*e.g.,* by looking at the specific code changes associated to a pull request) and categorize them into common themes (*e.g.,* threshold-based mechanisms, usage of low-power mode); (iii) all researchers collaboratively organize the themes into a coherent set of distinguishable tactics via several iterations; and (iv) each identified tactic is carefully reported according to the tactics template established in [46]. Out of the 97 initial data points, 87 were about energy *awareness* and 10 data points were about energy *efficiency*. The result of phase 4 is the set of four architectural tactics for energy-efficient robotics software described in Section IV-A.

We opt for a systematic methodology for RQ1 because we want to be reasonably confident about capturing a representative set of architectural tactics. This approach addresses threats to validity related to the fact that there are not many data points to mine for architectural tactics (97 data points) Not applying a systematic approach might lead to having extremely noisy data in Phase 4 (which is purely qualitative) and/or missing relevant data points. Similar approaches have been followed in other domains [52][58].

C. Empirically Evaluating the Green Tactics (RQ2)

Implementation of the robotics system – For this experiment we use a ROBOTIS TurtleBot3 [11], a two-wheeled ground robot. We use the Turtlebot because it is (i) one of the most popular robots for education and research [1], (ii) open source (both hardware and software) and thus highly customizable, (iii) affordable and available worldwide, (iv) small sized, and (v) designed to be fully compatible with ROS. In the context of this study, we use the following sensors: (i) a ROBOTIS LDS-01 LIDAR [3], (ii) an SDK ICM-20648 Inertial Measurement Unit (IMU) including a 3-axis gyroscope, accelerometer, and digital motion processor [6], and (iii) a 5-megapixel camera supporting different frame rates (FPS) [4]. The robot is powered by a Lithium polymer 11.1V 1800mAh battery.

The application layer of the robotic system is a customization of the default implementation provided by ROBOTIS [12]; it includes 6 ROS nodes (*e.g.,* one for the low-level control of the robot, one for managing the stream of data produced by the LIDAR, one for the camera), 6 ROS topics (*e.g.,* for velocity commands, odometry information), and a variable number of ROS services, depending on the applied green tactic. In our study the Turtlebot performs always the same mission: to move at 0.6m/s within a dedicated robotic arena while (i) continuously video-recording the mission at 60FPS and (ii) stopping every 20s and doing a 360°rotation at 0.8rad/s. The mission and the software implementing it are designed so to be (i) representative of classical robotics scenarios (*i.e.,* environment exploration), (ii) easily customizable, and (iii) simple enough to foster independent replication and verification.

Experimental variables – The independent variables are selected based on the goal of the experiment, to mitigate the mono-operation bias as much as possible (due for example to the usage of a single movement strategy), and to keep the experiment feasible in terms of execution time. The independent variables are described below.

- Tactic: the applied tactic. It has six treatments: (i) in the *baseline (b)* no green tactic is applied, (ii) the *EE1*, *EE2*, *EE3* and *EE4* treatments represent the system where the corresponding tactic is applied, and (iii) *combined (c)* represents the system where all four tactics are applied.
- Movement: the movement strategy of the robot. It has three treatments: (i) *noMovement*, where the position of the robot is fixed, (ii) *autonomous*, where the robot moves within the arena in a pseudo-random fashion, and (iii) *sweep*, where the robot executes a grid-based coverage path plan in the arena [33]. Movements (ii) and (iii) have obstacle avoidance capabilities.
- Environment: represents whether the robotic arena has obstacles or not. It has two treatments: *empty*, *cluttered*.

The dependent variable is the total *energy* consumed by the robot during the mission. Energy values are computed by following a sampling-based approach [17], [36], [37], [64], that is: (i) sampling the power consumption of the robot at a fixed interval, (ii) applying the $E = P \times t$ formula, where P= measured power and t = 120 seconds (*i.e.,* the duration of our mission), and (iii) solving the integral of P over t. We measure the power samples via the technique proposed by Hindle *et al.* for mobile apps [37]. We mount an Arduino NANO [5] onboard the robot that measures voltage, current, and power (in mW) drawn from the Turtlebot battery via a INA219 current sensor [40] (sampling rate = 200Hz). This solution allows us to precisely measure the power consumption of the robot with minimal impact on the measurement process. In order to not interfere with the robot under measurement, the Arduino NANO persists the energy measures to an SD card. Due to space limitations, we do not provide the details about the circuitry, code, and auxiliary hardware of our measurement infrastructure, they are available in the replication package.

Experiment design – Based on the experimental variables and hypothesis, the experiment follows a 6x3x2 full factorial design, *i.e.,* all possible combinations of treatments across all independent variables. However, we do not consider the six *noMovement-cluttered* combinations because in those cases the presence of obstacles does not influence the execution of the mission. This leads to a total of 30 trials for our experiment, *i.e.,* (6x3x2)-6 combinations.

Experiment execution – The experiment is carried out in a 4.5m x 3.5m robotic arena where missions can be executed without interruptions and under the same conditions. Both the base station and the robot run on the same WiFi network they are the only devices connected to the network. We keep the execution environment as clean as possible by having a clean installation of the Raspbian OS on the robot, configuring both the robot and the base station to not perform any OS updates, disabling all services not needed for mission execution, etc.

Each trial of the experiment is repeated 10 times with a 2-minute idle wait between each run in order to account for possible fluctuations of the measured energy consumption and the *tail energy* phenomenon, where components such as the network card remain in high power states after completing a task [54]. As a result, the experiment is composed of 300 individual runs for a total of 600 minutes (10 hours) of sheer robot time. We randomize execution order of the experiment runs to avoid potential unpredictable confounding factors.

Each run of the experiment consists of six steps: (1) equip the robot with a fully-charged battery so that each run starts with the same battery level, (2) place the robot in its start position and rearrange the robot arena based on the *environment* factor, (3) bring up the robotic mission, (4) start the mission, (5) (after termination of the mission – 120 seconds), read and sanity check the measures persisted to the SD card, and (6) reset and reconfigure the system for the next run.

Data analysis – Firstly, we explore the collected energy measures via violin plots and summary statistics. Then, we analyze the distribution of the energy measures in order to check if a parametric test (*e.g.,* the one-way ANOVA) can be applied, which can potentially lead to higher statistical power w.r.t. non-parametric tests [74]. However, the energy measures across tactics are not normally distributed, even after applying several data transformations [71], [56]).

We apply the Kruskal-Wallis test (with $\alpha = 0.05$), a rank-based non-parametric test for testing whether two or more samples all come from identical populations [44]. In the context of our study, we use the Kruskal-Wallis to determine if there are statistically significant differences of energy consumption for every treatment of the *tactic* variable. The *magnitude* of the difference of energy consumption is estimated via the Eta-squared statistic and interpreted according to [69].

In order to identify which tactics lead to significantly different energy consumption, we perform a pairwise comparison between each tactic and the baseline using the Wilcoxon test [38] with Benjamini-Hochberg correction [68]. The comparison is carried out both globally and across all possible combinations of movement strategy and physical environment.

We assess the *magnitude* of the difference of each tactic via the Cliff's Delta effect size measure [27]. The values of the Cliff Delta measures are interpreted according to [34].

IV. Results

A. *Tactics for Energy-Efficient Robotics Software (RQ1)*

The green tactics identified in this study share the common goal of saving the energy consumed by a robot. Each tactic description below includes the number of its occurrences, the motivation for using the tactic, a component-and-connector (C&C) view that shows the main components of the tactic (see Figure 2), a description of the tactic based on the C&C view, and an example of how it is used in one of the data points considered in this study. Tactics are intended to be blueprints for implementation. Roboticists make decisions regarding the concrete implementation of the different components and connectors based on their system requirements and constraints.

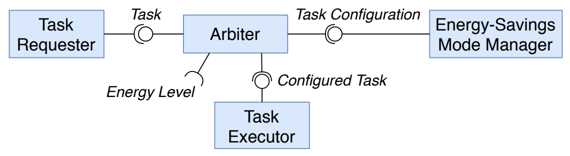
EE1: Limit Task (5). *Context* – Robot tasks such as streaming large videos or robot navigation can consume significant amounts of energy. Therefore, limiting these tasks when a robot reaches a critical energy level is important for extending the time that a robot is operational. Since several robotic platforms allow roboticists to query the current status of the battery (*e.g.,* in terms of charge, remaining life time, temperature), one way to limit execution of energy-hungry tasks is to place the robot in energy-savings mode once the energy level reaches an established threshold. For each of these tasks there is a default mode and an energy-savings mode, as shown in the examples in Table II.

TABLE II: Default and Energy-Saving Mode for Robot Tasks

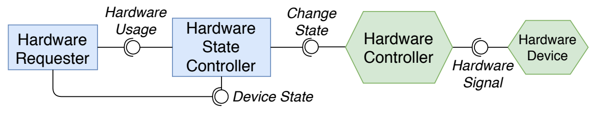
Default Mode	Energy Savings Mode
Move in any direction at the max power rate	Adjust power rate to 50% of set max power rate
Publish any type of data	Do not publish PCL point clouds
Send video stream to the operator display	Do not send any video streams

Solution – The *Limit Task* tactic configures a robot's task to execute in energy-savings mode when energy levels reach a given threshold (see Fig. 2(a)). The *Task Requester* requests to execute a task, the *Arbiter* decides whether to execute the task in default- or energy-savings mode, the *Energy-Savings Mode Manager* provides the task configuration for energy-savings mode, and the *Task Executor* executes the task, as such:

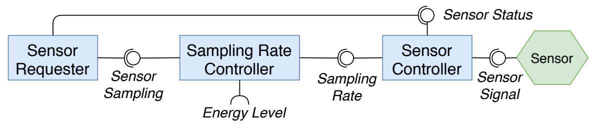
1) The *Task Requester* sends a task to the *Arbiter*.
2) After receiving the task, the *Arbiter* checks the energy level of the robot (provided by another component).
3) If energy level is below the established threshold, the *Arbiter* requests the energy-savings mode task configuration from the *Energy-Savings Mode Manager*.
4) The *Arbiter* forwards the received task to the *Task Executor* for execution.
5) The *Arbiter* continues checking the energy level during the execution of the task.
6) If the energy level is below the threshold, the *Arbiter* obtains the task configuration from the *Energy-Savings Mode Manager* and instructs the *Task Executor* to continue execution of the task in its energy-savings mode.
7) Similarly, if the energy level rises above the threshold, the *Arbiter* instructs the Task Executor to continue execution of the task in its default mode.
8) Once the task is completed the *Task Requester* is notified.



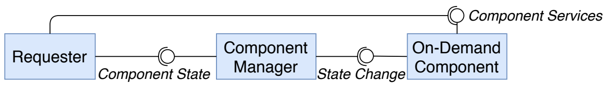
(a) *Limit Task* tactic (EE1)



(b) *Disable Hardware* tactic (EE2)



(c) *Energy-Aware Sampling* tactic (EE3)



(d) *On-Demand Components* tactic (EE4)

Fig. 2: C&C view for the identified green tactics

Example scenario – Data point 36 is about a ROS-based system that uses haptic devices (data point 36). Haptic teleoperation allows a user to perform manipulation tasks in distant, scaled, hazardous, or inaccessible environments [62]. In this system, the haptic device controller is a ROS node that communicates tasks to another ROS node which in turn controls the robot. When the energy of the robot arm reaches a critical level, the robot arm controller node adjusts the received task based on its configuration for energy-savings mode. In addition, because the haptic device controller is subscribed to an arm feedback topic, it can inform the user about the arm's battery state as an indication for why it is operating at a slower speed so that haptic user feedback can be adjusted.

EE2: Disable Hardware (3). *Context* – Hardware components (*e.g.,* servos, drivetrains) of a robot often consume a significant amount of energy. For example, in addition to the energy required to power a motor such as the Dynamixel XC430-W240 [63], the controller of the motor also consumes energy due to the CPU usage to process the produced data (*e.g.,* about its current velocity and temperature). It is therefore important to prevent unnecessary utilization of hardware resources in order to extend the operation time of the robot.

Solution – The *Disable Hardware* tactic disables hardware components when they are not strictly needed, which results in less energy consumption by the robot and more efficient power management (Figure 2(b)). The tactic is implemented to manage the state of the actual *Hardware Device*, as such:

1) The *Hardware Requester* notifies the *Hardware State Controller* whether or not the hardware device is needed for a certain task.
2) The *Hardware State Controller* instructs the *Hardware Controller* to disable or enable the *Hardware Device*.
3) Before enabling or disabling the *Hardware Device*, the *Hardware Controller* checks if it is safe to change the state of the HW device (*e.g.,* to toggle a hardware pin).
4) If it is safe, the Hardware Controller enables or disables the *Hardware Device*.
5) Note that it is also possible for the *Hardware Requester* to obtain the state of the *Hardware Device* at any time via the *Hardware State Controller*.

Example scenario – The *ros_control* package [9] is one of the most used ROS packages. In data point 23, the *controller_manager* node advertises two services: a *load_controller* service (enable hardware) and an *unload_controller* service (disable hardware). If a node needs to enable the robot hardware, it sends a request to the *load_controller* service. If it wishes to disable the robot hardware, it sends a request to the *unload_controller* service. After receiving a request, the *controller_manager* node performs the request by either enabling or disabling the robot hardware. The requesting node is notified with the result to ensure that it is aware of the robot hardware status.

EE3: Energy-Aware Sampling (1). *Context* – In robotics, many sensors are designed to provide a continuous stream of data (*e.g.,* accelerometers, LIDARs, cameras) [21]. However, sampling data from sensors consumes energy, especially as incoming data is continuously processed (CPU usage).

Solution – The *Energy-Aware Sampling* tactic shown in Figure 2(c) adjusts the rates for sensor sampling based on the energy level of the robot, as such:

1) The *Sensor Requester* asks the *Sampling Rate Controller* to start sampling the *Sensor* at a given rate.
2) The *Sampling Rate Controller* instructs the *Sensor Controller* to start sampling the *Sensor* at the given rate and continues checking the energy level during the execution of the sampling task.
3) If the energy level reaches a critical threshold, the *Sampling Rate Controller* instructs the *Sensor Controller* to start sampling at a lower rate and informs the *Sensor Requester* of the adjusted sampling rate.
4) It is also possible for the *Sensor Requester* to obtain sensor status at any time from the *Sensor Controller*.

Example scenario – Data point 51 is about the ROS-based driver for InvenSense's 3-axis gyroscope [2]; there, an MPU controller node subscribes to a *battery_state* topic to check the battery level and a *sampling_rates* topic to which sensor sampling rates are published by an MPU node. Based on battery levels, the MPU Controller adjusts sensor sampling rates accordingly by sending a request to the sampling action advertised by the MPU node that controls the actual sensor.

EE4: On-Demand Components (1). *Context* – Continuously running a ROS node requires the spawning of an operating system (OS) process which is an energy-consuming task in terms of CPU/memory usage (*i.e.,* executing a CPU-intensive loop) and other resources (*e.g.,* sensors, motors, fans for cooling). Therefore, it is necessary to ensure that OS processes are not running if they are not needed.

Solution – The *On-Demand Components* tactic shown in Figure 2(d) starts new components only when their functionality is needed. The *Requester* represents a component that requires the functionality of the *On-Demand Component*. The *Component Manager* acts as a controller that either starts up or shuts down a component based on requests, as such:

1) The Requester indicates to the *Component Manager* that it needs the *On-Demand Component* to be in either the online or offline state.
2) The *Component Manager* starts up (online) or shuts down (offline) the *On-Demand Component* depending on its status and the number of clients already using its functionality.
3) The *Component Manager* notifies the *Requester* of the state of the *On-Demand Component*.
4) If the *On-Demand Component* is online, the *Requester* can start using its services.
5) Once the *Requester* no longer requires the functionality of the *On-Demand Component* it goes back to Step 1) to change its state to offline.

Example scenario – In data point 14, in order for the camera to operate, it requires the *camera_driver* ROS nodelet to be up and running. The requester nodelet publishes the required state for the camera (online/offline) to a *camera_status* topic, which is subscribed to by the nodelet manager. Based on the published required status, the nodelet manager either starts up or shuts down the *camera_driver*. Once the *camera_driver* is up, it advertises a service that can be called by the requester.

Note that tactics EE4 and EE2 are related, but different. The main difference is that on-demand components (EE4) also covers software-only cases (*e.g.,* an intermediate node for acting as message broker), whereas EE2 focuses specifically on hardware resources (and their states).

B. Empirical Evaluation of the Tactics (RQ2)

For this study we implement the tactics as minimally as possible so to (i) avoid the interaction between tactics implementation and unnecessary confounding factors (*e.g.,* the communication overhead due to messages exchanged between additional components), (ii) ease the replication of the experiment, and (iii) facilitate the comprehension of each tactic. We implement the green tactics into our Turtlebot as follows:

- EE1: limit the movement of the robot by waiting 5 seconds before each 360°rotation;
- EE2: disable the camera of the robot (*i.e.,* with no video acquisition) when the robot is moving among locations;
- EE3: lower the frame rate of the camera to 30 FPS;
- EE4: kill the ROS node of the camera when the robot is moving and bring it up before each 360°rotation.

As discussed in Section III-C, our experiment also includes a *baseline* where no tactics are applied and a *combined* treatment where all the tactics are applied simultaneously.

Data exploration. The energy consumption across all tactics ranges between 1067.08 and 1429.11 Joules (see Table III), with a median (mean) of 1277.74 (1271.80) Joules. The standard deviation of the collected energy measures is non negligible and ranges from 51.70 for the EE1 tactic to 72.78 for the EE3 tactic; overall, the values of the standard deviation are mainly due to the robot performing different movements during the execution of the mission and to the intrinsic fluctuation of energy and it justifies our design choice of repeating the runs of the experiment 10 times for each trial. Nevertheless, the coefficient of variation remains between 4% and 6%, making us reasonably confident about the reliability of the measurement infrastructure we setup for the experiment.

TABLE III: Descriptive statistics of the energy consumption in Joules (SD=standard deviation, CV=coefficient of variation)

Tactic	Min.	Max.	Median	Mean	SD	CV
Baseline (B)	1151.93	1416.81	1336.96	1318.11	60.92	4.62
EE1	1164.58	1386.37	1293.06	1291.62	51.70	4.00
EE2	1089.45	1369.67	1258.91	1255.11	62.44	4.97
EE3	1130.56	1429.11	1337.52	1313.29	72.78	5.54
EE4	1084.92	1321.92	1250.00	1239.13	63.67	5.14
Combined (C)	1067.08	1322.60	1225.36	1213.56	59.18	4.88
Global	1067.08	1429.11	1277.74	1271.80	72.77	5.72

Result 1 – On *average* the application of green tactics tend to improve energy-efficiency, however not all tactics impact the energy consumption of the system with the same magnitude.

As shown in Figure 3, tactic EE4 is the one impacting energy the most, making the Turtlebot consume an average of 78.55 Joules less than the baseline treatment, followed by EE2 and EE1 with an average saving of 63.0 and 26.49 Joules, respectively. The EE3 tactic shows a slightly different behaviour; even though on average it saves 4.82 Joules, when it is applied the system tends to consume the same (or even more) energy w.r.t. the baseline (the median energy consumption of EE3 is 0.56 Joules *higher* w.r.t. the baseline). This result may seem surprising, however it can be explained by the way we implemented the EE3 tactic. Indeed, EE3 just changes the frame rate of the sensor to 30 FPS but the Turtlebot does not use the acquired video stream, *e.g.,* by persisting, manipulating, or streaming the recorded video. We decided to implement EE3 in this way so to completely isolate the application of the tactic from the business logic managing the data produced by the camera. In summary, in the specific context of our experiment the application of EE3 did not lead to energy savings (the is also statistically confirmed). This phenomenon is also confirmed in the mobile apps domain [61], where lowering the frame rate of a camera does not impact its energy consumption *per se*, rather energy is impacted the most by *how* the recorded video stream *is used* in other components of the system (*e.g.,* streaming the recorded video to the cloud).

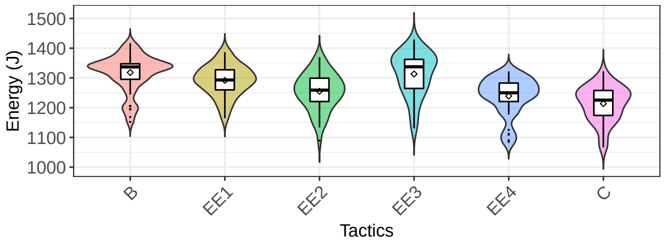


Fig. 3: Energy consumption across architectural tactics (B=baseline, EEi=ith applied tactic, C=combined, ◇=mean)

These results are statistically confirmed, with the Kruskal-Wallis test producing a p-value of 2.74×10^{-18} (with *large* effect size), which allows us to reject the null hypothesis that the energy measures at each tactic come from identical populations [66]. The pairwise comparison between each tactic and the baseline with the Wilcoxon test further confirm our results; the p-value for EE1, EE2, and EE4 is lower than 4×10^{-3}, thus rejecting the null hypothesis that the median difference between the baseline-EE1, baseline-EE2, and baseline-EE4 pairs is zero. We find a *medium* effect size for EE1 (0.34) and a *large* effect size for EE2 and EE4. These results provide evidence about the fact that the application of the EE1, EE2, and EE4 tactics lead to a significantly different amount of energy consumption in the context of our experiment.

Result 2 – The combination of all green tactics improves energy-efficiency more than each tactic in isolation.

The median (mean) energy consumed by the Turtlebot with all combined tactics is 111.6 (104.55) Joules less than the baseline. This difference is far higher than those related to the individual tactics: the application of the tactics leads to a 7.9% energy saving on average. To put this result into perspective, considering that the total energy of the battery of the Turtlebot is 71928 Joules and that on average our 2-minute missions consume 1271.8 Joules, the total lifetime of a Turtlebot without tactics is about 109 minutes, whereas the application of the tactics would like to a total lifetime of about 119 minutes (a 10-minute improvement over a mission of less than 2 hours). The previously mentioned Wilcoxon tests statistically confirm this result with a p-value of 5.75×10^{-11} and the Cliff's delta measure reveal a *large* effect size (0.78).

Result 3 – The movement strategy and the physical environment influence how energy is consumed during the mission.

Figure 4 shows the power measurements collected during one randomly-chosen mission for each combination of movement strategy and physical environment. Among others, here we can clearly notice (i) the generally lower power consumption of the robot with the applied tactics with respect to the baseline, (ii) the low power consumption of the robot with the *noMovement* strategy in the first 20 seconds of the mission, where the robot still does not use at all the wheel actuators (Figure 4a), and (iii) the more chaotic power consumption in the *cluttered* environment due to the avoidance of the encountered obstacles (Figures 4c and 4e).

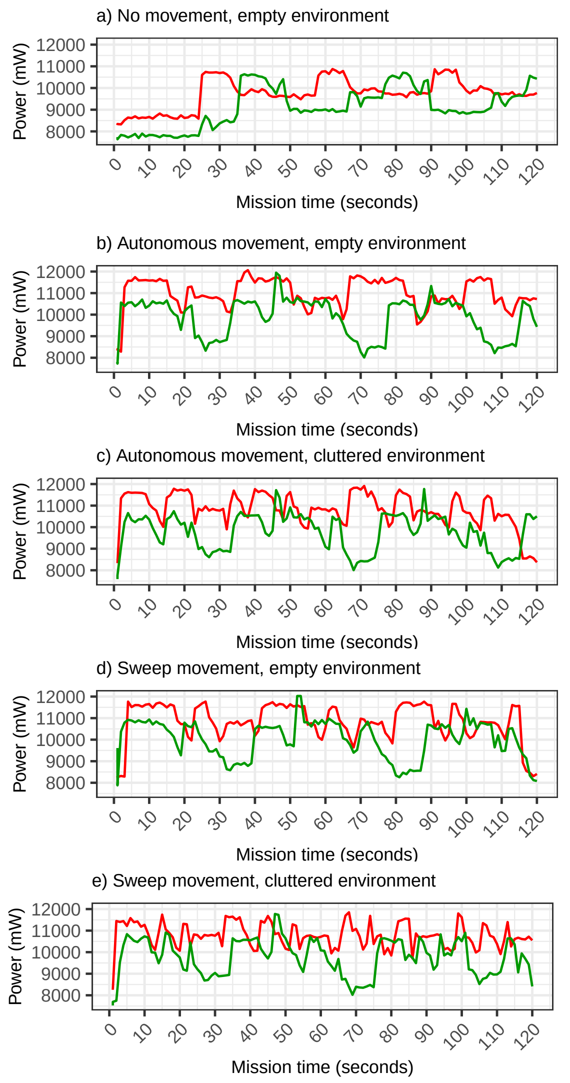


Fig. 4: Examples of power measurements across all movements and environments (baseline, combined)

By looking at the combinations of tactic, movement strategy, and physical environment (see Figure 5), we can witness that different amounts of energy are consumed when the robot is moving. This result is expected since additional energy is consumed by the two actuators for rotating the wheels of the robot. More interestingly, we can also confirm the general results obtained when discussing results 1 and 2. Specifically, almost all tactic-movement-environment combinations lead to a statistically significant difference in terms of energy consumption, with the exception of EE3. Moreover, when the results are statistically significant, their effect size is always *large*. This gives evidence about the fact that applying tactics EE1, EE2, and EE4 (and their combination) likely leads to higher energy-efficiency, with a *large* effect on it.

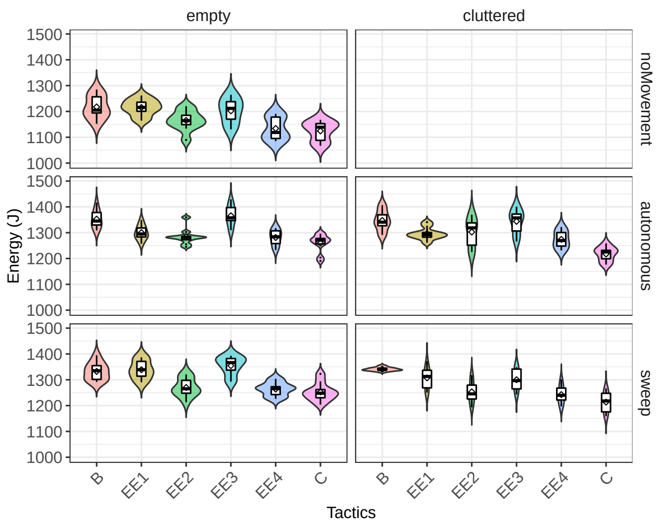


Fig. 5: Energy consumption across all movements strategies and environments

We have three other exceptions to the main trend, namely the *EE1-noMovement-empty*, *EE1-sweep-empty*, and *EE2-autonomous-cluttered* combinations. About the *EE1-noMovement-empty* combination, we speculate that it is due to the fact that movement is a fundamental component of EE1, thus having the robot standing still for the whole duration of the mission (the *noMovement* treatment) would have penalized the EE1 tactic. At the time of writing we do not have hard evidence for explaining the results about the other two combinations since they all involve different combinations of factors. Further analysis and replications of the experiment are already planned for clarifying this specific part of the study.

V. Discussion

Energy is infrequently discussed by roboticists. Of the 339,563 ROS data points only 562 (0.17%) mention energy-related topics. This result is quite counter-intuitive, considering that (i) the majority of the discussed systems involve battery-powered mobile robots and (ii) the lifetime of current battery-operated robots is low. This phenomenon might also relate to the lack of testing/debugging tools for energy-efficient or at least energy-aware robotics software (*e.g.,* accessible energy measurement tools, libraries for energy-aware programming for robots); concern also confirmed in other domains, *e.g.,* mobile apps [57], where two main problems are developer awareness and lack of tools. We do not have hard evidence explaining whether the infrequence of energy-related data points is due to lack of awareness, tools, or simply interest by roboticists. A follow-up qualitative study might shed light on this phenomenon. Our answer to RQ2 empirically demonstrates that the software design choices of roboticists do impact the energy-efficiency of robots, often with large effects. This should motivate researchers on energy-efficiency to focus on robotics software, and developers to adopt the green tactics we identified and seek (and document) new ones.

Roboticists tend to not document architecture. The advantages of architecture documentation are widely reported [26], including facilitating the onboarding of newcomers to open-source projects, and being able to discuss/reason about trade-offs between system-level quality attributes like energy and performance. However, none of the 97 analyzed data points include a documented architecture, *e.g.,* via a diagram or a thorough description of the involved components, connectors, and their configuration; finding also confirmed by another study on the architecture of ROS-based systems [48]. We suggest roboticists to document the architecture of the (part of) system they want to discuss in order to better clarify their general points, design decisions, and rationale to the reader of their posts in discussion and social coding platforms and code/comments in their own source code.

There are other green tactics out there. The green tactics we identified are not meant to be complete. We designed our study so to let the green tactics *emerge* from the practice; there may be other sources for the tactics, like robotics textbooks, interviews with robotics experts, grey literature. Our choice is motivated by two main forces: (i) to empirically assess how and to what extent *practitioners* deal with energy-efficiency at the architectural level, and (ii) to focus on tactics that are applied in real contexts. In principle, the latter point makes the green tactics directly applicable in real projects. The identified tactics provide evidence of how roboticists achieve energy efficiency by having tradeoffs with respect to other robot functionalities. Also, the quantitative evidence provided in RQ2 help in promoting a more careful treatment of energy-related aspects of robotics software and the tradeoffs and related side-effects for the specific application domain, thus leading to better robotics software in general.

The green tactics should be refined and used in context. The tactics should be used and refined depend on the specific context of the system being developed. For example, the main component of tactic EE1 (*i.e.,* Limit Task) is the Arbiter, which decides whether to execute a task in default or energy-saving mode. However, there are many different types of tasks (*e.g.,* paint a wall, drive to a point of interest) and many definitions of modes depending on the specific robot at hand (see Table II). Thus, roboticists must understand the context-sensitive tradeoffs implied by the system under development and apply the tactics accordingly. This observation is specially true when considering *performance* and *maintainability* since the presented tactics can involve having additional tactic-specific components/roles (*e.g.,* the Arbiter in EE1, the Component Manager in EE4), potentially leading to (i) communication and computation overhead and (ii) higher complexity of the system, thus hindering future improvements over time.

Know the Physics of your robot. One of the lessons learned during the execution of our experiment is that sensors and actuators might behave in counter-intuitive ways from the perspective of software developers. For example, the first implementation of tactic EE1 consisted in limiting the robot to 30% of its nominal speed; the intuition was that slower robots would make less "work" than faster robots, thus saving energy. Some pilot runs showed that this assumption was completely wrong. Indeed, the electric motors for rotating the wheel of the Turtlebot actually was consuming more energy at slower speeds! This is mainly due to the fact that the majority of the input power at slower speeds was used to overcome the dynamic friction inside the motors and as the speed was increasing, friction played a smaller and smaller role in their overall efficiency [63], [13]. We suggest researchers and roboticists to have the energy-related behaviour of their robots under control by (i) carefully studying the technical specifications of all hardware components of the robots and, based on that, (ii) benchmarking the energy consumption of their robots under different conditions and configurations.

VI. Threats to Validity

External validity. The data sources for answering RQ1 might not be representative of the complete state of the practice on robotic systems, *e.g.,* not all types of robots might be covered or roboticists might discuss their technical concerns using other platforms. This potential threat to validity is mitigated by considering multiple (heterogeneous) data sources while building the dataset and by mining all the contents of social discussion platforms (*e.g.,* ROS Answers) in full. Moreover, the mined open-source repositories underwent a strict search and selection process [48], making us reasonably confident of their representativeness (*e.g.,* no toy or demo projects).

The mission we implemented for answering RQ2 might not be representative of all possible robotic missions. Also, different missions might obviously change the energy consumption, or even the tactics. This threat is mainly due to the feasibility of the performed experiment and we mitigated it by (i) using one of the most used robots within the ROS community (*i.e.,* the Turtlebot), and (ii) performing tasks which are very common in the robotics domain, such as navigating within an environment and avoiding obstacles. This study focuses on ROS-based systems and therefore the identified tactics might not apply to other types of systems. However, given that tactics by definition are described at a high level of abstraction in order to support multiple implementations, we are reasonably confident that they can also apply to non ROS-based systems.

Internal validity. The qualitative analysis for answering RQ1 could lead to subjective results, specially because it involved the manual categorization of several heterogeneous data points. This potential threat is mitigated by involving three researchers in phases 1, 2, and 3 and by always jointly discussing conflicts, with one of the researchers acting as arbiter.

When answering RQ2, history and maturation threats [74] are mitigated by (i) using three different batteries and always fully charging and alternating the one used in each run, (ii) executing every run at 2-minute intervals, (iii) rebooting the robot and clearing its execution environment between runs, and (iv) carefully checking the status of the hardware at every run. We further mitigated this threat by setting up a minimal and replicable measurement infrastructure (see Section III-C). Also, we could assume that the noise produced by the execution environment of the mission is similar for all runs [16].

Construct validity. For RQ1, we are confident about the correct implementation of all data extractors for building the dataset (phase 1) because we carefully checked each of them in isolation via subsets of data for which we already knew the expected results. Their implementation is publicly available in the replication package. In phase 2 the main threat is related to the use of a pattern-based approach for identifying energy-related data points. We mitigated this potential threat by letting energy-related keywords emerge from previous empirical studies on software energy-efficiency. In Phase 3, the architectural data points are selected by systematically applying a set of selection criteria defined *a priori*, similarly to common practices in systematic literature studies [42].

For RQ2, we are reasonably confident about the correctness of the implementation of the green tactics because they were developed after all green tactics were fully defined and their source code underwent several reviews before running the experiment. The measured energy consumption could suffer from a systematic measurement bias. This threat is mitigated by the fact that the readings from the INA219 sensor have been retrieved redundantly; this allowed us to reconstruct the consumed energy for every run in multiple ways and to cross-check them against each other. Moreover, we repeated every trial of the experiment ten times to take into account possible fluctuations of the measured energy consumption [16].

Conclusion validity. For RQ1, other researchers might identify and organize the collected data points differently, thus potentially leading to different tactics. We mitigated this threat to validity by (i) carefully documenting the process we followed for all phases for the tactics identification (see Section III-B), (ii) having Phases 2, 3, and 4 collaboratively conducted by multiple researchers and by statistically analysing their level of agreement, and (iii) making the raw data produced in every phase available for independent verification.

For RQ2, we first checked if parametric statistical tests were applicable (in principle parametric tests are more powerful than non-parametric ones). The assumptions underlying the applied statistical tests were checked (*e.g.,* normality), and when their assumptions could not be met: (i) data was transformed in order to make it compatible with the required assumptions [71] and (ii) if the data was still incompatible, non-parametric tests were used. We are reasonably confident about the absence of fishing for certain results because we (i) corrected the obtained p-values when performing multiple statistical tests and (ii) did not remove any data from our experimental runs (*e.g.,* outliers).

VII. Related Work

Energy-Efficiency in Robotics Software. Mei *et al.* [50] present an approach for energy-efficient robot exploration. Their approach determines the next position for the robot to visit based upon orientation information and it is validated on a simulated robot. Licea *et al.* [47] focus on an algorithm for energy-efficient wireless communication. The robots visit different locations and share sensed data in a way that minimizes their energy consumption. Siar and Fakharian [65] use optimized control algorithms to minimize the energy consumption of arm robots. To validate the results, various simulations are carried out. Robots using Mecanum wheels [72] have a limited energy-budget so research on improving their energy-efficiency is abundant. For example, Xie *et al.* [75] incorporate an improved obstacle detection and avoidance algorithm to improve the energy-efficiency of Mecanum-wheeled robots. Our study complements the studies mentioned above since it focuses on energy-efficiency solutions (*i.e.,* our green tactics) defined *at the architectural level* instead of dealing with low-level algorithms, control strategies, and network optimizations.

Architectural Tactics for Energy-Efficiency. Studies have been conducted on green tactics for other domains, *e.g.,* for cyber-foraging [45] or cloud-based software [59], although missing empirical evaluation. A work similar to ours has been conducted for Android and iOS applications [29]: commits, issues, and pull requests from GitHub are inspected via a thematic analysis to identify energy patterns, like dark UI colors, (open/start resources only when strictly necessary (similar to our EE4 tactic), and increase time between syncs/sensors sampling (similar to our EE3 tactic). As future work we would like to conduct a thematic analysis on our dataset to complement our results with a discussion of recurring energy patterns. Kjærgaard and Kuhrmann [43] present a catalog of green tactics for mobile sensing, along with a preliminary validation. Some of their tactics are (i) provide sensor data in an on-demand manner and (ii) provide requested information in an on-demand manner (both similar to our tactic EE4).

VIII. Conclusions

Based on an extensive mining of the ROS software ecosystem, in this paper we identify and empirically evaluate a first body of architectural tactics for energy-efficient robotics software. The results show that (i) the green tactics significantly help improve the energy-efficiency of the robot and (ii) context and SW-HW interplay play an important role for their most-effective selection. Given the surprising lack of focus of the studied roboticists on energy-related topics, this green-tactics body of knowledge can help roboticists start changing their practice by (i) adopting these green tactics and (ii) becoming aware of the benefit of documenting and communicating their architecture design decisions, possibly leading to a new (energy-aware) development mindset. Further, the specific application domain or context (*e.g.,* pedestrian safety in driver-less automotive) influence significantly the tradeoffs one can or cannot make between energy-efficiency and *e.g.,* functionality. To ensure that mission-criticality is prioritized, future research should explore contextualizing the tactics for specific situations.

Acknowledgments

This research is partially supported by the Dutch Research Council (NWO) through the OCENW.XS2.038 grant.

References

[1] Top 10 ROS-based robotics companies to know in 2019. https://www.therobotreport.com/top-10-ros-based-robotics-companies-2019, Jul 2019. [Online; accessed 19. Aug. 2020].

[2] 3-Axis | TDK. https://invensense.tdk.com/products/motion-tracking/3-axis, Aug 2020. [Online; accessed 20. Aug. 2020].

[3] 360 Laser Distance Sensor LDS-01 (LIDAR). http://www.robotis.us/360-laser-distance-sensor-lds-01-lidar, Aug 2020. [Online; accessed 19. Aug. 2020].

[4] Camera Module - Raspberry Pi (version 1.3). https://www.raspberrypi.org/documentation/hardware/camera, Aug 2020. [Online; accessed 19. Aug. 2020].

[5] Getting Started with the Arduino Nano. https://www.arduino.cc/en/Guide/ArduinoNano, Aug 2020. [Online; accessed 19. Aug. 2020].

[6] ICM-20648 | TDK. https://invensense.tdk.com/products/motion-tracking/6-axis/icm-20648, Aug 2020. [Online; accessed 19. Aug. 2020].

[7] International Technology Roadmap for Semiconductors. http://www.itrs2.net/itrs-reports.html, Jul 2020. [Online; accessed 21. Jul. 2020].

[8] ROS Community Metrics. http://wiki.ros.org/Metrics, Jul 2020. [Online; accessed 28. Jul. 2020].

[9] ros_control - ROS Wiki. http://wiki.ros.org/ros_control, Aug 2020. [Online; accessed 20. Aug. 2020].

[10] TC Software Engineering for Robotics and Automation - IEEE Robotics and Automation Society. https://www.ieee-ras.org/technical-committees/146-technical-committees/software-engineering-for-robotics-and-automation, Aug 2020. [Online; accessed 28. Aug. 2020].

[11] TurtleBot 3 - ROBOTIS. http://www.robotis.us/turtlebot-3, Jul 2020. [Online; accessed 30. Jul. 2020].

[12] turtlebot3_bringup - ROS Wiki. http://wiki.ros.org/turtlebot3_bringup, Aug 2020. [Online; accessed 28. Aug. 2020].

[13] Why is an electric motor more efficient at higher loads? https://physics.stackexchange.com/questions/46113/why-is-an-electric-motor-more-efficient-at-higher-loads, Aug 2020. [Online; accessed 27. Aug. 2020].

[14] M. Albonico, I. Malavolta, G. Pinto, E. Guzmán, K. Chinnappan, and P. Lago. Mining energy-related practices in robotics software. In *Proceedings of the 18th International Conference on Mining Software Repositories, MSR*, page To appear, New York, NY, May 2021. ACM.

[15] Anonymous. Study Replication Package. https://github.com/S2-group/msr-2021-robotics-green-architectural-tactics-replication-package, 2021.

[16] L. Ardito, R. Coppola, M. Morisio, and M. Torchiano. Methodological guidelines for measuring energy consumption of software applications. *Scientific Programming*, 2019, 2019.

[17] A. Banerjee, L. K. Chong, C. Ballabriga, and A. Roychoudhury. Energypatch: Repairing resource leaks to improve energy-efficiency of android apps. *IEEE Transactions on Software Engineering*, 44(5):470–490, 2017.

[18] V. R. Basili and H. D. Rombach. The tame project: Towards improvement-oriented software environments. *IEEE Transactions on software engineering*, 14(6):758–773, 1988.

[19] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[20] G. Bavota. Mining unstructured data in software repositories: Current and future trends. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 1–12. IEEE, 2016.

[21] M. Beetz. *Plan-based control of robotic agents: improving the capabilities of autonomous robots*, volume 2554. Springer Science & Business Media, 2002.

[22] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[23] D. Brugali and E. Prassler. Software engineering for robotics [from the guest editors]. *IEEE Robotics & Automation Magazine*, 16(1):9–15, 2009.

[24] S. A. Chowdhury and A. Hindle. Characterizing energy-aware software projects: Are they different? In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 508–511, 2016.

[25] F. Ciccozzi, D. D. Ruscio, I. Malavolta, P. Pelliccione, and J. Tumova. Engineering the software of robotic systems. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 507–508. IEEE Press, May 2017.

[26] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting software architectures: views and beyond*. Pearson Education, 2002.

[27] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.

[28] J. Cohen. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, 70(4):213, 1968.

[29] L. Cruz and R. Abreu. Catalog of energy patterns for mobile applications. *Empirical Software Engineering*, 24(4):2209–2235, 2019.

[30] D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, Sep. 2011.

[31] P. Estefo, J. Simmonds, R. Robbes, and J. Fabry. The robot operating system: Package reuse and community dynamics. *Journal of Systems and Software*, 151:226–242, 2019.

[32] A. Fischer-Nielsen, Z. Fu, T. Su, and A. Wąsowski. The Forgotten Case of the Dependency Bugs. 2020.

[33] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[34] R. J. Grissom and J. J. Kim. *Effect sizes for research: A broad practical approach.* Lawrence Erlbaum Associates Publishers, 2005.

[35] E. H2020. Robotics 2020 Multi-Annual Roadmap For Robotics in Europe - http://sparc-robotics.eu/wp-content/uploads/2014/05/H2020-Robotics-Multi-Annual-Roadmap-ICT-2016.pdf. 2016.

[36] S. Hao, D. Li, W. G. Halfond, and R. Govindan. Estimating mobile application energy consumption using program analysis. In *2013 35th international conference on software engineering (ICSE)*, pages 92–101. IEEE, 2013.

[37] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 12–21, 2014.

[38] M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric statistical methods*, volume 751. John Wiley & Sons, 2013.

[39] G. J. Holzmann. Landing a spacecraft on mars. *IEEE software*, 30(2):83–86, 2013.

[40] A. Industries. INA219 High Side DC Current Sensor Breakout - 26V $\pm$3.2A Max. https://www.adafruit.com/product/904, Aug 2020. [Online; accessed 19. Aug. 2020].

[41] ISO. *ISO/IEC/IEEE 42010, Systems and software engineering — Architecture description*, December 2011.

[42] S. Keele et al. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.

[43] M. B. Kjærgaard and M. Kuhrmann. On architectural qualities and tactics for mobile sensing. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, pages 63–72, 2015.

[44] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

[45] G. Lewis and P. Lago. A catalog of architectural tactics for cyber-foraging. In *International ACM Conference on Quality of Software Architectures*, pages 53–62. ACM, 2015.

[46] G. Lewis and P. Lago. Architectural tactics for cyber-foraging: Results of a systematic literature review. *Journal of Systems and Software*, 107:158–186, 2015.

[47] D. B. Licea, D. McLernon, M. Ghogho, and S. A. R. Zaidi. An energy saving robot mobility diversity algorithm for wireless communications. In *21st European Signal Processing Conference (EUSIPCO 2013)*, pages 1–5, 2013.

[48] I. Malavolta, G. Lewis, B. Schmerl, P. Lago, and D. Garlan. How do you architect your robots? state of the practice and guidelines for ROS-based systems. In *ACM/IEEE International Conference on Software Engineering*, 2020.

[49] H. Malik, P. Zhao, and M. Godfrey. Going green: An exploratory analysis of energy-related questions. In *IEEE/ACM Working Conference on Mining Software Repositories*, pages 418–421, 2015.

[50] Y. Mei, Y.-H. Lu, C. S. G. Lee, and Y. C. Hu. Energy-efficient mobile robot exploration. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 505–511, 2006.

[51] D. Meike, M. Pellicciari, and G. Berselli. Energy efficient use of multirobot production lines in the automotive industry: Detailed system modeling and optimization. *IEEE Transactions on Automation Science and Engineering*, 11(3):798–809, 2013.

[52] I. Moura, G. Pinto, F. Ebert, and F. Castor. Mining energy-aware commits. *Working Conference on Mining Software Repositories*, 2015.

[53] M. Nagappan and E. Shihab. Future trends in software engineering research for mobile apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 21–32. IEEE, 2016.

[54] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168, 2011.

[55] R. Paulissen, S. Kalisingh, J. Scholtes, A. van Geldrop, and A.-L. Hoftijzer. Robotics in the Netherlands. *Netherlands Foreign Investment Agency (NFIA) report*, 2016.

[56] R. A. Peterson. Using the bestNormalize Package, Jun 2020. [Online; accessed 27. Aug. 2020].

[57] G. Pinto and F. Castor. Energy efficiency: a new concern for application software developers. *Communications of the ACM*, 60(12):68–75, 2017.

[58] G. Pinto, F. Castor, and Y. D. Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 22–31, 2014.

[59] G. Procaccianti, P. Lago, and G. A. Lewis. Green architectural tactics for the cloud. In *2014 IEEE/IFIP Conference on Software Architecture*, pages 41–44. IEEE, 2014.

[60] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[61] S. V. Rajaraman, M. Siekkinen, and M. A. Hoque. Energy consumption anatomy of live video streaming from a smartphone. In *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pages 2013–2017. IEEE, 2014.

[62] P. Rezeck, B. Frade, J. Soares, L. Pinto, F. Cadar, H. Azpurua, D. G. Macharet, L. Chaimowicz, G. Freitas, and M. F. Campos. Framework for haptic teleoperation of a remote robotic arm device. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, pages 170–175. IEEE, 2018.

[63] ROBOTIS. Dynamixel XC430-W240 manual. https://emanual.robotis.com/docs/en/dxl/x/xc430-w240, Aug 2020. [Online; accessed 20. Aug. 2020].

[64] E. A. Santos, C. McLean, C. Solinas, and A. Hindle. How does docker affect energy consumption? evaluating workloads in and out of docker containers. *Journal of Systems and Software*, 146:14–25, 2018.

[65] M. V. S. Siar and A. Fakharian. Energy efficiency in the robot arm using genetic algorithm. In *2018 8th Conference of AI & Robotics and 10th RoboCup Iranopen International Symposium (IRANOPEN)*, pages 14–20. IEEE, 2018.

[66] M. Sullivan and J. Verhoosel. *Statistics: Informed decisions using data*. Pearson Boston, MA, 2013.

[67] S. Swanborn and I. Malavolta. Energy efficiency in robotics software: A systematic literature review. In *35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '20)*, pages 137–144. ACM, 2020.

[68] D. Thissen, L. Steinberg, and D. Kuang. Quick and easy implementation of the benjamini-hochberg procedure for controlling the false positive rate in multiple comparisons. *Journal of educational and behavioral statistics*, 27(1):77–83, 2002.

[69] M. Tomczak and E. Tomczak. The need to report effect size estimates revisited. an overview of some recommended measures of effect size. *Trends in Sport Sciences*, 21(1):19 – 25, 2014.

[70] F. Ullah and M. A. Babar. Architectural tactics for big data cybersecurity analytics systems: A review. *Journal of Systems and Software*, 151:81–118, 2019.

[71] S. Vegas. Analyzing software engineering experiments: everything you always wanted to know but were afraid to ask. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 534–535, 2018.

[72] Wikipedia. Mecanum wheel. https://en.wikipedia.org/w/index.php?title=Mecanum_wheel&oldid=973708843, Aug 2020. [Online; accessed 26. Aug. 2020].

[73] T. Witte and M. Tichy. Checking consistency of robot software architectures in ros. In *2018 IEEE/ACM 1st International Workshop on Robotics Software Engineering (RoSE)*, pages 1–8, May 2018.

[74] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering*. Computer Science. Springer, 2012.

[75] L. Xie, C. Henkel, K. Stol, and W. Xu. Power-minimization and energy-reduction autonomous navigation of an omnidirectional mecanum robot via the dynamic window approach local trajectory planning. *International Journal of Advanced Robotic Systems*, 15(1):1729881418754563, 2018.