Agents of Autonomy: A Systematic Study of Robotics on Modern Hardware

MOHAMMAD BAKHSHALIPOUR, Carnegie Mellon University, USA
PHILLIP B. GIBBONS, Carnegie Mellon University, USA

As robots increasingly permeate modern society, it is crucial for the system and hardware research community to bridge its long-standing gap with robotics. This divide has persisted due to the lack of *(i)* a systematic performance evaluation of robotics on different computing platforms and *(ii)* a comprehensive, open-source, cross-platform benchmark suite.

To address these gaps, we present a systematic performance study of robotics on modern hardware and introduce *RoWild*, an open-source benchmark suite for robotics that is comprehensive and cross-platform. Our workloads encompass a broad range of robots, including driverless vehicles, pilotless drones, and stationary robotic arms, and we evaluate their performance on a spectrum of modern computing platforms, from low-end embedded CPUs to high-end server-grade GPUs. The source code of the benchmark suite is available in `https://cmu-roboarch.github.io/rowild/`.

Our findings reveal that current architectures experience significant inefficiencies when executing robotic workloads, highlighting the need for architectural advancements that satisfy the primary requirements of robotic tasks. We discuss approaches for meeting these requirements, offering insights for improving the performance of robotics.

CCS Concepts: • **Computing methodologies** → **Robotic planning**; **Modeling methodologies**; • **General and reference** → **Measurement**; **Evaluation**; **Performance**; • **Computer systems organization** → *Real-time system architecture*; **Architectures**.

Additional Key Words and Phrases: Robotics, Computer Architecture, Benchmarking

ACM Reference Format:
Mohammad Bakhshalipour and Phillip B. Gibbons. 2023. Agents of Autonomy: A Systematic Study of Robotics on Modern Hardware. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 3, Article 43 (December 2023), 30 pages. https://doi.org/10.1145/3626774

1 INTRODUCTION

The advancement of robotics technology is rapidly changing the world we live in. With predictions of 20 million robots by 2030 [3] and a market capitalization of US$210 billion by 2025 [171], it is clear that robotics will play an increasingly important role in society. To become widespread, robots need to meet the demands of real-world environments, which necessitates them being autonomous and capable of performing complex artificial intelligence (AI) tasks in real-time [124].

Computer hardware and architecture play a paramount role in realizing real-time robotics, evidenced by the deployment of robot-specific hardware accelerators in the architecture of the latest edge processors. Recent robotic platforms [81, 82, 109, 113] include hardware accelerators

Authors' addresses: Mohammad Bakhshalipour, bakhshalipour@cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; Phillip B. Gibbons, gibbons@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

2476-1249/2023/12-ART43
https://doi.org/10.1145/3626774

for operations like tree-extension and ray-casting that have massive usage in robotics; Intel's multi-robot system [81] has a full-fledged "Robot SoC."

Surprisingly, the computer system and architecture research community has under-explored robotics. Despite the ever-growing importance of robots in our technological society, as well as the surge of industrial "robotic processors," there is a large gap between robotics and the computer systems research community, borne out by the scant few publications.[1] The gap deprives robotics of many improvements achievable by system-level techniques, as we will show in this paper.

The gap is largely because of the lack of *(i)* a systematic performance study and *(ii)* a comprehensive, open-source, cross-platform benchmark suite. As a result of *(i)*, the robotic tasks, their performance requirements, and their system-level implications are unclear to the community. And, due to *(ii)*, the few research papers include only one [60, 129, 142] or a couple of applications [87, 91, 158] in their evaluations, leaving the impact on other applications unknown.

This paper aims to bridge this gap by introducing *RoWild*[2] , a comprehensive, open-source, cross-platform robotic benchmark suite. The challenge of benchmarking robotics lies in the vast array of applications, from self-driving cars to home-assistant robots, with more to come in the future. It is impractical to represent all of these applications in a single benchmark suite. *RoWild* overcomes this challenge by exploiting the fact that different robots, despite their different applications, perform a finite set of common "robotic tasks" such as scene understanding and pathfinding. For instance, both self-driving cars and home-assistant robots require scene understanding. Nevertheless, the algorithms and constraints in conducting such tasks can vary widely across different applications.

RoWild comprises a wide range of individual robotic tasks, encompassing the software pipeline of practically all autonomous robots (§3). With versatility in mind, *RoWild* implements each task with various algorithms and parameters. This flexible approach enables the configuration and pipelining of tasks to model the end-to-end computation of diverse robotic applications. By including a broad set of tasks and algorithms, *RoWild* is capable of modeling numerous robotic applications, thus providing a comprehensive benchmark suite.

Our choice of implemented tasks and algorithms stems from an analysis of 29 industrial robots, encompassing a diverse range from arm manipulators and home-cleaning robots to self-driving vehicles. This analysis was undertaken to ensure a broad yet relevant selection. Additionally, we also incorporate state-of-the-art research algorithms (e.g., deep learning-based pathfinding) into *RoWild*, positioning it as a suitable suite for future robotics. We develop *RoWild* with essential considerations that distinguish it from previous work, making it suitable for systems research. Specifically, *RoWild* is high-performance, simulator-friendly, versatile, and modular.

This paper's second contribution is to investigate the system-level implications of robotics (§4). Using *RoWild*'s tasks, we model six different end-to-end robotic applications and evaluate them on a spectrum of platforms, ranging from low-end embedded CPUs to high-end server-grade GPUs. While previous studies [66, 127, 181] have conducted some system-level analysis (e.g., CPU vs. GPU) specific to their applications, these analyses remained at a high level. In contrast, our study delves deeper to investigate low-level implications, including the efficacy of caching, prefetching, and vectorization. This research is the first of its kind in the realm of robotics.

Our system-level investigations reveal *significant inefficiencies in the architecture* of today's prevailing compute platforms when executing robotic workloads (§5). Specifically, we find:

[1] In 2022, out of the tens of thousands of papers published in the top systems conferences (according to `csrankings` categorization, e.g., computer architecture, HPC, computer networks), only a handful were related to robotics.

[2] The name RoWild is derived from "Robotics in the Wild," signifying our focus on evaluating robotic performance in natural and unpredictable settings.

- **Vectorization Is Ineffective.** Despite the large silicon real-estate it occupies, CPU vectorization does little for robotic tasks: in the common case of an axis-unaligned orientation, the robot's memory layout is not vectorization-friendly.
- **On-Edge Parallelism Hits The Memory Wall.** Massive parallelism on edge platforms (e.g., Nvidia's Jetson Nano) is bottlenecked by memory: the memory wall is hit well before Amdahl's law.
- **Simple Prefetchers Are Inadequate.** While simple data prefetchers can help with robotics workloads, complex AI algorithms used in robotics use irregular data structures, producing memory patterns that defeat commercial prefetchers.
- **Caches Perform Significant Unnecessary Data Movements.** Hardware caches are unaware of the robots' software semantics; in many cases, caches work *in opposition* to them. As a consequence, caches perform excessive data movements, utilizing the memory hierarchy inefficiently.

We discuss potential solutions to address these inefficiencies and unveil untapped opportunities.

2 PRELIMINARIES AND RELATED WORK

2.1 The Software Pipeline of Robots

Fig. 1 shows the software pipeline of a generic autonomous robot. It consists of three stages: *perception*, *planning*, and *control*.

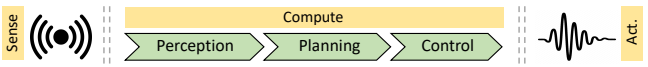


Fig. 1. The operation model of a generic autonomous robot.

The perception stage is responsible for comprehending the state of the robot and its surrounding environment. It reads raw sensory data and infers the robot's state (e.g., position, orientation) and the surrounding environment (e.g., obstacles around the robot). Inferring the state of the robot and the environment are called *localization* and *mapping*, respectively.

The planning stage uses the perception stage's output and finds an efficient (e.g., short), collision-free path from the robot's current position toward the goal position.

The control stage generates commands for the robot's actuators based on kinematics and dynamics, such as velocity and torque, to enable efficient execution of the planned path.

2.2 Terminology

Below, we define some of the terminology we employ throughout this paper:

- **Robotic Tasks:** The term "robotic tasks" pertains to *generic* high-level operations such as scene understanding and pathfinding. The range of these operations is finite within the field of robotics, but the ways they are carried out can vary from one robot to another. For instance, both autonomous vehicles and home-assistant robots undertake the task of pathfinding. However, the employed algorithms and the constraints could significantly differ in these two scenarios.
- **Robotic Algorithms:** By "robotic algorithms," we refer to specific algorithms implemented to carry out robotic tasks; e.g., the A* algorithm [154] for pathfinding. There are hundreds, or even thousands, of such algorithms proposed in the robotics community to accomplish robotic tasks.
- **Robotic Applications:** The term "robotic applications" denotes the end-to-end software pipeline of a specific robotic system, for instance, an autonomous drone. The scope of robotic applications is vast, extending from self-driving vehicles to home-assistant robots, and the spectrum continues to widen with future advancements in robotics.

2.3 Prior Work

Recent research [66, 67, 96, 181, 184] has highlighted the crucial role that robots' compute capabilities play in their overall performance, energy efficiency, and safety, dispelling the myth that the importance of computation in robots is secondary to their mechanical components. For instance, PerceptIn, a startup that specializes in autonomous vehicles, has reported that a staggering 88% of the operation latency of their vehicles is attributed to the compute system [181]. Similarly, in a study conducted by Boroujerdian et al. [67], it was found that enhancing the compute platform (e.g., increasing the number of cores from two to four) resulted in a three-fold improvement in the flight time and energy consumption of autonomous drones.

Spurred by these findings, recent systems research proposes design methodologies [68, 118, 145–147] and hardware acceleration [60, 134, 142, 166] to achieve efficient, real-time robotics.

Several prior works [66, 127, 181] report the characteristics of a *particular* application of robotics. For instance, Lin et al. [127] and Yu et al. [181] study autonomous driving, and MAVBench [66] studies micro aerial vehicles. Each of these studies targets a particular application of robotics, leaving out many others (e.g., stationary arms, humanoid robots).

Similarly, prior work proposes benchmark suites [50, 72, 105] and studies the efficacy of different algorithms [73, 85, 96] for *particular* robotic tasks. For instance, OMPL [50] is a suite for sampling-based planning, and RLBench [105] is a suite for robot learning. Each of these suites targets a specific task, not the entire robotic software pipeline. Also, combining these suites into a broad set of workloads is not straightforward, because they use unalike setups (e.g., C++ vs. Python).

ROS [46] is a robotic middleware, offering codes for package management, device control, and software libraries. Nonetheless, ROS does not consider performance as the main objective: over three-quarters of the codes are written in Python, and even those written in C++ are not all high-performance. Further, the use of Python and TCP-based inter-process communication primitives in ROS conflicts with many microarchitectural simulators, such as [75, 78, 104, 137, 155].

Table 1. Feature comparison of related work and RoWild. More ✔ is better.

Paper/Repository	Scope	End-to-End	High-Perf.	Simulator-Friendly	Multi-Platform	Versatile & Modular	Open-Source	System-Level Analysis
Lin et al. [127] Yu et al. [181]	**Self-Driving Cars**	✔	✔✔	Unknown	✔✔	Unknown	✘	✔
MAVBench [66]	**Drones**	✔	✔	✣	✔	✘	✔	✔
One-off [50, 72, 105]	**Single Task**	✘	✣	✣	✘	✔	✔	✘
ROS [46]	**Broad**	✘	✣	✘	✘	✔	✔	✘
Educational [13, 42]	**Broad**	✘	✘	✣	✘	✘	✔	✘
RTRBench [61]	**Broad**	✘	✔	✔	✘	✔	✔	✘
RoWild	**Broad**	✔	✔✔	✔	✔✔	✔	✔	✔✔

✣ Depending on the case (e.g., kernel, simulator), it can be ✘ or ✔.

Educational repositories, such as [13, 42], implement various robotic tasks. Nevertheless, the performance of these repositories are orders of magnitude far from real-time [61]. Our prior work RTRBench [61], a code collection of 16 robotic algorithms, featured some improvements in performance over previous suites; however, it is single-threaded and lacks essential features, such as end-to-end application modeling, cross-platform compatibility, and detailed system-level evaluations. *RoWild* extends RTRBench, rectifying these limitations and establishing itself as an apt benchmark suite for exploring the nexus between robotics and systems.

Table 1 puts *RoWild* in the context of prior work. In §3, we detail how *RoWild* overcomes the shortcomings of prior work, establishing it as a highly-valuable robotic suite for systems research.

3 THE ROWILD APPROACH

3.1 Workloads

To model a variety of robotic applications, *RoWild* implements a rich set of individual robotic tasks and algorithms. Table 2 lists these tasks and the algorithms implemented to execute them, with symbols indicating the real-world robots that either come pre-packaged with the software where the algorithm is used or are custom-programmed in the literature to use the algorithm to perform a specific mission.[3]

Stage	Task	Algorithm(s) in RoWild
Perception	State Estimation	**MCL**$^{\dagger\ddagger\Upsilon\P\eta\sigma}$ [183], AMCL$^{\zeta}$ [172]
	Localization and Mapping	**EKF**$^{\Upsilon\S\tau\phi\Lambda}$ [173], Fast$^{\ddagger\P\xi}$ [139], Graph$^{\vartheta\varphi}$ [164], ORB$^{\varsigma\varrho}$ [74]
	Scene Reconstruction	**Point-Based Fusion**$^{\dagger\ddagger\Upsilon\kappa\upsilon}$ [178]
	Grid Generation	**Probabilistic Occupancy Map**$^{\dagger\kappa\vartheta}$ [94]
	Object Detection	**MobileNet-SSD**$^{\ddagger\Upsilon\varrho\chi\Lambda}$ [180]
Planning	2D/3D Navigation	**WA**$^{\star\dagger\P\zeta\eta\alpha\Pi\Lambda}$ [154], **GA*** [186], **RA*** [62], IDA$^{\star\varsigma}$ [115] , DQN [138]
	Timed Search	WA$^{\star\dagger\P\zeta\eta\alpha\Pi\Lambda}$ [154] + Backward Dijkstra [63]
	2D/3D *n*-DoF Search	**RRT**$^{\dagger\ddagger\Upsilon\kappa}$ [93], RRT$^{\star\Upsilon\omega\gamma\delta}$ [95], PRM$^{\Upsilon\vartheta\psi}$ [168], Shortcut [97]
	Reactive Planning	**Behavior Tree**$^{\dagger\lambda\vartheta\phi}$ [98]
	Task Scheduling	Symbolic Planning$^{\S\vartheta\delta}$ [64]
Ctrl.	Motion Control	**MPC**$^{\Upsilon\Pi\vartheta\dagger\kappa\alpha\psi\varepsilon}$ [89] **PID**$^{\Upsilon\vartheta\zeta\theta\phi}$ [177], **PP**$^{\ddagger\Upsilon\kappa\phi}$ [148]
	Parameter Learning	**DMP**$^{\Upsilon\vartheta\xi\alpha\theta\iota}$ [116], CEM$^{\S\lambda}$ [151], BO$^{\varrho}$ [106]

Table 2. RoWild's workloads. Algorithms in **color** are characterized in this paper in the context of end-to-end robotic applications.

Real-World Robots:

†Spot [8]
‡DJI Phantom [38]
$^{\Upsilon}$LoCoBot [2]
§Atlas [7]
¶AMR [43]
$^{\Pi}$AscTec Pelican [6]
$^{\zeta}$Roomba 980 [44]
$^{\upsilon}$Roomba i7+ [45]
$^{\eta}$Husky [18]
$^{\vartheta}$YuMi [58]
$^{\kappa}$Jackal [25]
$^{\xi}$LBR [30]
$^{\varrho}$Pepper [48]
$^{\sigma}$TurtleBot [55]
$^{\varsigma}$TurtleBot3 [56]
$^{\tau}$MiR [31]
$^{\phi}$AG [28]
$^{\Lambda}$Pioneer 3-DX [40]
$^{\varphi}$UR10e [52]
$^{\chi}$TALOS [49]
$^{\psi}$KR60-3 [29]
$^{\omega}$Grizzly [12]
$^{\gamma}$Skydio [47]
$^{\delta}$M-200iA/2300 [15]
$^{\varepsilon}$PerceptIn [180]
$^{\alpha}$Boxbot [9]
$^{\theta}$UR5e [53]
$^{\iota}$Franka Panda [16]
$^{\lambda}$PAL TIAGo [54]

Our selection of algorithms is the result of an exhaustive examination of the 29 real-world robots listed in Table 2 (symbols), ensuring that they represent a broad majority of robots employed in practical scenarios. We incorporate algorithms that *(i)* are applicable to real-time environments, *(ii)* have proven efficacy within the robotics community, and *(iii)* enable *RoWild* to emulate a wide and diverse array of end-to-end robotic applications (see §4).

Importantly, while *RoWild* emphasizes state-of-the-art algorithms, it also includes seminal algorithms that have stood the test of time and remain highly relevant in robotics. This is aligned with the interests of system manufacturers who strive to accelerate these algorithms. For example, [81], [113], and [82] fabricate hardware accelerators for the classic EKF [172], IDA* [115], and RRT algorithms [121], respectively.

RoWild uses these tasks to model various robotic applications. For each robotic application modeled, *RoWild* selects the necessary tasks, *specializes* each task to suit the application's requirements (see §3.2), and integrates them into a pipeline to model the end-to-end functionality of the robot.

3.2 Key Features and Considerations

Our development of *RoWild* involves essential considerations that make it suitable for systems and hardware research, as outlined below:

[3]In some cases, the algorithms are identified based on the demystification of other works. However, it should not be assumed that the robots listed are certainly/exclusively using these algorithms.

Comprehensive: As discussed in §3.1, *RoWild* includes a diverse range of robotic workloads. This feature is crucial from a computer architecture perspective; it is unlikely that hardware vendors will fabricate a specific processor for every robotic application.

High-Performance: Performance is at the heart of *RoWild*'s design, setting it apart from other open-source robotic repositories that face issues like Python overhead [46, 105, 159] and a lack of modern software techniques [13].

To optimize performance, we develop codes from scratch in native languages and use industry-standard profilers [20, 36] to identify execution bottlenecks and focus on accelerating them.

We make use of various high-performance software techniques including `constexpr` branches and `constexpr` functions that enable the compiler to perform calculations at compile time, thereby improving execution by eliminating not-taken branches. We employ built-in functions provided by GCC for aligning and prefetching data, providing branch prediction hints, and maximizing loop unrolling to enable efficient out-of-order execution. Additionally, we make use of the high-performance VCL [4] for manual code vectorization.

Simulator-Friendly: Because most hardware research uses simulators, we develop *RoWild* to be compatible with existing architectural simulators (e.g., no Python runtime, no TCP-based inter-process communications as in ROS). By default, *RoWild* codes are integrated with zsim [161].

Cross-Platform: *RoWild* caters to a wide range of compute resources by developing applications in C++, CUDA, and Verilog. This is particularly relevant for modern AI/robotic platforms that come equipped with heterogeneous compute resources [21, 22, 32–35, 37, 51, 57].

Versatile: *RoWild* is a versatile repository that not only comprises diverse robotic workloads but also develops each of them in a configurable manner. This configurability empowers users to perform each task using a range of algorithms and parameters, enabling the study of realistic robots that feature specialized software components for specific robotic applications and environments.

RoWild is designed to seamlessly integrate with a wide variety of sensor models and actuation systems. It means that whether a robot employs optical sensors for vision, acoustic sensors for auditory perception, or tactile sensors to detect physical contact, *RoWild* is able to interface with it. In terms of movement and actions, *RoWild* is compatible with diverse actuators, from traditional electric motors to pneumatic systems and even cutting-edge actuation mechanisms.

Modular: *RoWild* embodies modularity in its design, enabling the seamless pipelining of different tasks to model end-to-end robotic applications. In each of the applications exemplified in §4, the integration of the various tasks into the pipeline required fewer than 20 lines of C++ code.

Open-Source: *RoWild* is distributed under the MIT License and is available as open-source software.

3.3 Compute Platforms

Various compute platforms can be used in robotics. For example, safety-critical self-driving cars might use server-grade, high-end GPUs; on the other hand, low-cost surveillance robots might use affordable, low-end CPUs. *RoWild* evaluates a range of processors, as shown in Table 3 and detailed below:

Table 3. The evaluated compute platforms (October 2023 prices).

Acronym	Platform	Cores	Freq. (GHz)	TDP (W)	Memory (GB)	Price (US $)
LC	ARM Cortex A57 CPU	4	1.43	10	4	149
LG	Nvidia Maxwell GPU	128	0.92	10	4	149
HC	Intel Xeon Gold CPU	20 (×2)	2.10	125	384	1493
HG	Nvidia Titan X GPU	3584	1.41	250	12	999

***LC*:** A low-end, quad-core ARM Cortex A57 CPU available on Nvidia's Jetson Nano board [27]. It uses a 4 GB LPDDR4 memory with up to 25.6 GB/s bandwidth.

***LG*:** A low-end, 128-core GPU with Maxwell architecture available on Nvidia's Jetson Nano board [27]. It shares the 4 GB LPDDR4 memory system with the ARM CPU.

***HC*:** A high-end, server-grade Intel Xeon Gold 5218R CPU [24] with 20 two-way cores. It uses six 64 GB DDR4 memory modules, each with up to 38 GB/s bandwidth.

***HG*:** A high-end, super-clocked Nvidia Titan X GPU [14] with 3072 CUDA cores featuring the Pascal architecture. It has a 12 GB GDDR5 memory with up to 336 GB/s bandwidth.

3.4 Software Workflow and Validation

We develop CPU codes with C++17. We compile programs with GCC 11 with `-O3` and run them under Ubuntu 22.04. We develop GPU codes with CUDA 11 and compile them with NVCC. We set the number of thread blocks and threads-per-block empirically to obtain the fastest execution time for every application.

To model the producer-consumer execution model (i.e., pipelining the tasks), we use C++'s asynchronous execution primitives. Tasks run on separate threads (host or device), and producer-consumer pairs communicate through shared `future` objects. This way, different tasks are effectively pipelined, maximizing resource utilization.

3.4.1 Verification. We rigorously verify *RoWild*. We cross-check the end-to-end results with other repositories [1, 13, 42] and mathematical expectations (e.g., confirming the final path is the shortest). Furthermore, *RoWild* operates in two modes: *verification mode* and *high-performance mode*. The verification mode incorporates extensive, fine-grained assertions to confirm correctness. This feature will assist future researchers in modifying the programs and ensuring that their changes do not lead to errors. In the high-performance mode, these assertions are removed to expedite execution.

3.4.2 Performance Validation. Because many factors (e.g., inputset, environment, configuration) affect the robotic workloads' performance, we refrain from making exact performance comparisons with other suites, and leave such task to potential users. Nevertheless, our measurements of *RoWild* (as detailed in §4) show that it is 1–3 orders of magnitude faster than the reported performance of *open-source* repositories like RTRBench [61], PythonRobotics [42], CppRobotics [13], and ROS [46] (and by extension, those that use them [66, 147]). These repositories suffer from the high overheads of Python runtime [42, 105], network communications (publisher-subscriber communication model in ROS [46]), and unoptimized software (e.g., single-threaded code [13, 42, 46, 61, 105], spinner callbacks [46], poor memory management [13]).

Compared to *open-source* real-world robots (e.g., [2, 41, 55]), *RoWild* offers significantly superior performance. Such robots typically are developed with objectives such as ease of programming in mind, and frequently utilize software not necessarily optimized for real-time performance. For example, running LoCoBot [2] with its pre-packaged `pyrobot` results in planning times exceeding seconds, while *RoWild* delivers millisecond-scale planning on identical hardware (see §4.3).

Finally, drawing direct comparisons between *RoWild* and industrial-grade real-time robots (e.g., [15, 28, 58]) is not viable due to the undisclosed specifics of the hardware and software utilized in these robots. However, the scattered data we have been able to gather suggest that *RoWild*'s performance is competitive. It aligns closely with the publicly reported performance numbers of PerceptIn [181], as well as performance numbers measured by other researchers across multiple real-world robots [132].

3.5 Measurements

We measure the performance (p) by how inversely proportional it is to the execution time (t); i.e., $p \propto \frac{1}{t}$. After running each program multiple times, we calculate the average execution time based on wall clock time. p quantifies how efficiently a robotic system operates by measuring the speed at which tasks are completed, making it a valuable tool for assessing performance and making direct comparisons between different approaches. In robotics, where real-time responsiveness is often crucial (see §6.1), this metric is particularly essential as it reflects a system's ability to process information and make decisions swiftly.

We do not conduct detailed power measurements since compute is *not* a significant consumer of power in robots [66, 67, 117]. In robots, over 95% of the power is consumed by the mechanical components, rendering computation's role in power (not energy) trivial. For example, in 3DR Solo Drone [5], rotors consume over 286W, while computation consumes less than 13W [66]. When computation's power consumption is negligible (i.e., $Power_{Total} \approx Power_{Rotors}$), the overall energy-efficiency is proportional to the computation *performance* (i.e., $Energy = Power \times t$).

4 THE MODELED APPLICATIONS

With the rich set of robotic tasks and algorithms that *RoWild* provides (§3), users can model a broad spectrum of robotic applications. In this paper, we make a concerted effort to model and study a *substantial and representative* range of robotic applications. Detailed in Table 4 are the modeled robotic applications, with corresponding references to analogous industrial robots[4] in terms of algorithms and missions, and the tested environments.

Table 4. The modeled end-to-end applications.

Name	Mission	Resembling	Environment
DeliBot	Delivery	Spot [8]	Our Campus
PatrolBot	Patrolling	Pioneer 3-DX [40]	Our Campus
MoveBot	Manipulation	LoCoBot [2]	Synthetic
HomeBot	Cleaning	Roomba i7+ [45]	Hypersim [156]
FlyBot	Photography	AscTec Pelican [6]	FR Campus [17]
CarriBot	Transportation	Boxbot [9]	Intel Lab [23]

Importantly, the robots we model are not only diverse in their applications, ranging from home assistance to campus patrolling, they also demonstrate a variety of computational behaviors. More specifically, we model robots that experience performance bottlenecks at *different stages of the software pipeline*. This offers two significant benefits to the hardware and systems community:

- It allows the examination of the effects of different techniques across a broader field of robotics. In recent years, several research proposals have targeted one specific robotic pipeline stage (e.g., perception [96, 181], planning [60, 166], and control [158]) and modified the architecture accordingly. While this benefits the specific robotic application studied (e.g., self-driving cars), the impact of such architectural changes has remained unknown for other robotic applications (e.g., home-assistant robots). By providing a benchmark with robots bottlenecked at different pipeline stages, we clarify such impacts.
- Identifying architectural bottlenecks *shared* across robots with differing computational behaviors and devising efficient hardware solutions makes for a more compelling case for adoption of the solutions by industry than addressing bottlenecks arising only in one or two similar robots.

[4]This is not to say that they utilize identical software. According to data from our industry partners, industrial-grade real-time robots operate on proprietary software infrastructures. These infrastructures, even when bearing core similarities to *RoWild* such as CUDA usage, diverge in facets like library usage. Given the proprietary nature of these systems, specifics are undisclosed and remain out of reach for the broader research community. In contrast, *RoWild* is developed using open-source software, with the goals of maximizing performance, closely emulating real-world robots, and maintaining public availability. The same holds true for the robots' hardware. Many robots do not disclose information about their processors and hardware peripherals for reasons ranging from cyber-attack prevention to competitive edge maintenance.

Below, we describe the modeled robots, analyze their computational behaviors, and report the average execution time for *one complete cycle of the software pipeline*, from sensor reading to actuation. This includes the inter-stage communication overheads, though they are typically negligible. Finally, notice while we evaluated the applications in specific environments, they are adaptable to a wide range of other environments.

4.1 DeliBot: Legged Robot Delivering Items

DeliBot is a legged robot capable of transporting loads. It navigates CMU's Wean Hall [11] and delivers items to various locations. As the robot already has a map of the building, it does not need to perform any mapping. Fig. 2.a shows an overview of the application.

4.1.1 Setting. Accurate localization is vital for DeliBot to navigate effectively through the building while avoiding obstacles. The robot uses a *laser rangefinder* and an *odometer* to measure distances to obstacles (blue arrows in Fig. 2.a) and track its distance traveled (red arrow in Fig. 2.a), respectively. These sensors work in tandem to provide data for position estimation, allowing the robot to navigate reliably in the building.

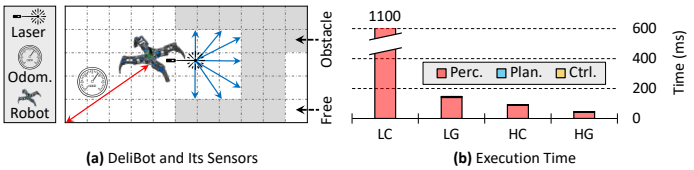


Fig. 2. DeliBot operating in our campus building.

DeliBot uses Monte Carlo Localization (MCL) [183], a.k.a. particle filter, to accurately determine its location and orientation within the building. MCL is a robust technique that can handle various sensor models (e.g., non-linear, non-Gaussian), which makes it suitable for complex environments like the building's narrow corridors.

MCL maintains multiple *particles*, each representing a particular hypothesis about the robot's state. Initially, all particles are assigned random beliefs, implying that the robot can be located anywhere in the building. As the robot moves and collects sensory data, MCL continuously *resamples* the particles: particles whose beliefs do not match sensory data are replaced with those that do. Over time, the beliefs of the particles converge towards the actual state of the robot.

We fine-tune MCL's parameters (e.g., the number of particles) such that it achieves a localization error—the difference between the robot's estimated and actual location—of a few centimeters or less, which is satisfactory for the application.

Because the environment is known and structured, DeliBot's planning is relatively simple, especially as the application does not require an optimal path. The robot simply selects a set of waypoints and navigates between them using a greedy search algorithm that directs it towards the nearest one.

Similarly, DeliBot's control stage is straightforward, with the controller simply selecting the leg to pull the robot forward. DeliBot's limited dynamics, such as fixed velocity and acceleration, contribute to the ease of control.

4.1.2 Evaluation. Fig. 2.b shows DeliBot's compute time on different platforms, where the planning and control stages are processed only on CPUs, while the perception stage leverages GPUs when available. The perception stage is the main contributor to the end-to-end latency, taking more than 97% of execution time. To ensure high localization accuracy in complex environments like the evaluated building, maintaining numerous particles in MCL is required. This improves accuracy by offsetting the effect of noisy measurements during the resampling process (§4.1.1), but it also results in a higher computational workload, making perception the computational bottleneck.

Fortunately, particles are independent and can be processed in parallel, which GPUs exploit to offer significantly higher performance compared to CPUs. Additionally, as particles converge and their hypotheses align, their control flow in response to sensor readings becomes similar, reducing branch divergence to near-zero, helping GPUs to offer superior performance during the steady stage. Comparing end-to-end times, *LG*/*HG* outperforms *LC*/*HC* by 7.9×/2.1×.

Ray-casting is the dominant operation performed by particles, accounting for 58%–83% of perception latency. It is performed in order to correspond laser measurements with particles' hypotheses and involves a cell-by-cell traversal of the environment map in different directions. Intel fabricates a "ray-casting hardware" [109] to accelerate this process.

Perhaps counter-intuitively, we find CPU vectorization ineffective for ray-casting. The reason being the generation of axes-unaligned memory accesses (blue arrows in Fig. 2.a) due to the *varying orientation* of the robot with respect to the environment axes, which current vectorization engines cannot handle. Further details on this observation are provided in §5.1.

Finally, as DeliBot employs a simple planner and controller, their impact on the end-to-end execution time is insignificant (up to 2.3% of execution time).

On the memory front, when the application is effectively parallelized, the memory bandwidth consumption can become substantial. On the low-end platforms, this reaches a peak bandwidth utilization of 4 GB/s, which is significant (see §5.2). On the high-end platforms, while the absolute memory bandwidth consumption is even greater due to increased parallelism, the relative impact on performance is less pronounced due to the expansive bandwidth resources inherent to the hardware.

Takeaways

• DeliBot's perception stage is a bottleneck, as it requires costly ray-casting operations within hundreds of particles to achieve acceptable localization accuracy in challenging environments, such as the narrow corridors of our building.

• LC's limited support for parallelism results in poor performance, making it unsuitable for time-critical applications with high accuracy demands.

4.1.3 Breadth. DeliBot's computation is designed to provide precise localization for robots operating in harsh environments. Examples of such robots include autonomous vehicles navigating through wilderness terrain [125] and agriculture robots used for crop monitoring and harvesting in challenging conditions [108].

4.2 PatrolBot: Wheeled Robot Patrolling Campus

PatrolBot is a wheeled robot that is responsible for patrolling our campus and detecting suspicious objects. Due to the dynamic nature of the environment, PatrolBot does not rely on a static map to navigate. Instead, it performs simultaneous localization and mapping (SLAM), which involves creating a map of the environment as it moves around and determining its own position in that map. To accomplish this task, PatrolBot uses six identified landmarks throughout the campus. An overview of the modeled application is shown in Fig. 3.a.

4.2.1 Setting. PatrolBot is equipped with *range* and *angle* sensors, which provide distance and angle measurements relative to the landmarks, respectively. PatrolBot uses these measurements as input to the Extended Kalman Filter (EKF) algorithm [173] to solve the SLAM problem.

EKF is a popular SLAM algorithm handling various sensors and providing accurate estimates of the robot's position and the environment's map, despite noise and uncertainties. It linearizes non-linear motion and measurement models, enabling a Gaussian probability distribution to represent the

system's state. This allows recursive state estimation using the Bayes' rule, which is computationally efficient and capable of handling uncertainties in the measurements.

While performing SLAM, PatrolBot also analyzes images from the surrounding area to detect suspicious objects. To accomplish this, it uses MobileNet-SSD [180], a lightweight neural network (NN) feature extraction algorithm, pre-trained on the MS COCO dataset [128]. MobileNet-SSD predicts object bounding boxes and classes using classifiers, and it is optimized for mobile devices with high accuracy. MS COCO is a widely-used, large-scale dataset with over 80 object categories, commonly used for image recognition.

If the NN cannot confidently categorize an object close to the robot or if it detects weaponry, PatrolBot infers that the object may be suspicious. If the object remains in the robot's view for a certain time, PatrolBot alerts the security personnel.

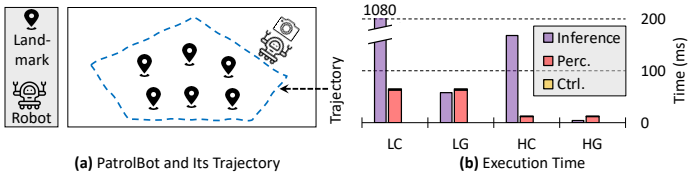


Fig. 3. PatrolBot operating in our campus.

PatrolBot follows a pre-determined path around the landmarks, without the need for online path planning. It utilizes the Pure Pursuit algorithm [148] to determine the optimal steering angle and speed for efficiently following the pre-determined path. Also, PatrolBot is equipped to detect obstacles, and it automatically stops and resumes its movements accordingly to avoid collisions.

4.2.2 Evaluation. Fig. 3.b shows PatrolBot' compute time across platforms. Notice, NN inference runs concurrently with the robot's software pipeline, not as a part of it. Therefore, the robot's overall performance is determined by the greater of the pipeline latency or the NN inference latency. The software pipeline runs entirely on CPUs, while the NN inference runs on GPUs when available.

On CPUs, NN inference determines the robot's performance. On GPUs, however, the inference is significantly accelerated, leading to the performance bottleneck being shifted to robot perception. GPUs run NN inference 6.5×–336× faster than CPUs, due to their massively-parallel architectures.

The perception stage, which runs EKF, is the second major compute component of PatrolBot. EKF heavily utilizes linear algebra, for instance, in computing the Jacobian matrix of state variables and propagating the covariance matrix to depict state uncertainty. The size of matrices and accordingly EKF's execution time scale with increasing the number of measurements; with six landmarks and two measurements per landmark in our setup, EKF becomes moderately time-consuming.

Finally, Pure Pursuit is relatively simple and fast, taking up to 3.2% of the end-to-end execution time. It calculates the robot's steering angle, which only requires computationally simple geometric calculations such as distance and angle calculations.

Takeaways

• Multiple compute platforms greatly improve a robot's real-time performance. LG, priced at $149, outperforms HC, priced at $1493, by running the NN on GPU and the rest on CPU. However, without the GPU (i.e., LC), it lags significantly.

4.2.3 Breadth. PatrolBot's computation represents robots that perform significant work beyond their software pipeline. Examples include security [185] and condition/environmental monitoring [107] robots, which continuously analyze sensory data to detect changes or potential hazards.

4.3 MoveBot: Arm Manipulator Moving Items

MoveBot is a manipulator equipped with a 5-degree-of-freedom (5-DoF) robotic arm that excels at moving objects with speed and precision. Its real-world applications are primarily in the manufacturing and warehousing industries, where products need to be moved quickly and efficiently. We evaluate MoveBot in a cluttered, synthetic environment, depicted in Fig. 4.a. We carefully model MoveBot after our in-house, open-source LoCoBot [2].

4.3.1 Setting. The environment is mostly static: the robot's base and the obstacles (walls) remain stationary. However, items are relocated by the robot itself. Hence, the robot performs perception only once offline, and keeps track of the moved objects to avoid collisions at runtime.

MoveBot is repeatedly queried to pick and place items. Upon a query, it performs path planning to move its end-effector to the designated position while avoiding obstacles. The planning is done in the joint angle space of the arm, which has five links. The planner finds a sequence of angles $(\alpha_1, \alpha_2, \ldots, \alpha_5)$ that moves the end-effector towards the goal position. Since the configuration space is high-dimensional, MoveBot's planner *samples* the space to find a path in a reasonable time.

Rapidly-exploring Random Trees (RRT) [120] is a popular algorithm for high-dimensional planning. Chung et al. [82] fabricate a full-fledged RRT-based path planning processor in 40 nm CMOS. RRT builds a tree from the start point to the goal by randomly sampling the configuration space and connecting the samples to the nearest nodes in the tree, ensuring that the path is collision-free. This requires numerous *collision detection* operations to check whether the robot will collide with obstacles as it moves from one configuration to another.

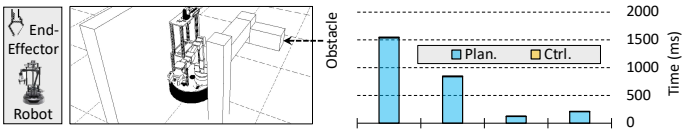


Fig. 4. MoveBot operating in a synthetic environment.

Accurate collision detection is a time-consuming process [61, 62, 65]. While hardware accelerators have been proposed [60, 126, 141–143], they are not widely adopted yet. On the software side, "reworked" methods have been developed that bound obstacles in the environment with simpler shapes like cubes and exploit their geometric properties to speed up collision detection [10, 88, 149, 176]. While these methods underestimate free space, they offer far faster performance. For MoveBot, we use the *cuboid-cuboid collision detection (CCCD)* algorithm [176], which bounds obstacles and the robot body with cubes and checks for intersection during movement planning. We find that CCCD strikes a good balance between accuracy and execution time for MoveBot.

Finally, MoveBot employs a simple PID controller [177] to move its arm by taking desired joint positions as input and generating control signals to the motors based on the difference between desired and actual joint positions.

4.3.2 Evaluation. Fig. 4.b shows MoveBot's compute time on different platforms, where the control stage runs on CPUs, while the planning stage leverages GPUs when available.

Collision detection, even in its reworked form, remains the most time-consuming aspect of planning, taking more than 85% of the end-to-end time. Each collision detection involves checking the intersection of four cubes around the robot's body (joints and peripherals) with nine cubes surrounding obstacles. *LC* offers limited parallelism support, resulting in the lowest performance, while *HC* utilizes both coarse-grained and fine-grained parallelism to achieve high performance. Coarse-grained parallelism occurs when different cuboid-cuboid checks are performed in parallel using different threads. Fine-grained parallelism happens because *HC*'s large instruction window

can extract high instruction-level parallelism (ILP) from operations like finding multiple *independent* neighbors at each RRT iteration.

We use Bialkowski et al.'s approach [65] to parallelize collision detections on GPUs, which we find to be more effective than other methods such as [102, 150]. This improves the performance of collision detection, especially on *HG*, which has larger on-chip caches and more cores. However, significant branch divergence occurs during collision detection. Our profiling with NVIDIA Nsight Compute [36] reveals that over 18% of branches are divergent, which is considered high [112]. This is due to CCCD conducting *multiple* checks (e.g., edges, normals, axes) on every configuration to determine collision status, and the outcome of checks differ, resulting in taking divergent flows. As the configurations are dynamically shaped at runtime, warp-synchronous programming is not applicable, and further improvements require dynamic techniques.

Finally, PID controller is simple and fast, taking up to 1.6% of execution time. It uses a linear combination of three terms, proportional, integral, and derivative, to calculate the control signal. Each term needs only basic arithmetic operations, such as subtraction, making the algorithm computationally cheap.

Robotic arm manipulators' memory demands can become substantial when they requires a large number of drawn samples. This often arises when the planner necessitates a *fine resolution* to ensure safe and accurate navigation in densely packed settings. Additionally, as DoF increase, memory consumption grows exponentially—a phenomenon termed the "curse of dimensionality." Even a minor uptick in DoF can drastically amplify the potential configurations the planner must account for, thereby escalating memory demands.

However, for the given application model and evaluation setting, the memory system is not the primary constraint. In the evaluated platforms, the 5-DoF robot operating within the tested environment does not significantly tax the memory bandwidth, with the exception of *LG*. On *LG*, memory is communally allocated between the host and the device. When collision detection undergoes massive parallelization on the GPU, the surge in memory requests places substantial strain on the bandwidth. This results in a notable performance drop, as elaborated in §5.2.

Takeaways

• Collision detection is costly and can pose a risk to real-time constraints, especially when running on low-end platforms. Safety-critical applications use a separate path to throttle robots directly in case of real-time collision detection failure [181].

4.3.3 Breadth. MoveBot's computation represents robots whose performance is bottlenecked by collision detection. This includes most arm manipulators with a high DoF, which find applications in domains like automotive manufacturing (e.g. welding [175]) and construction (e.g. bricklaying [133]).

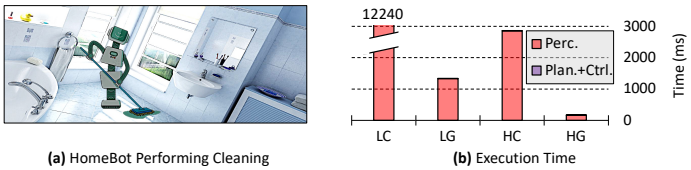
4.4 HomeBot: Assistant Robot Cleaning House

HomeBot is a home-assistant robot performing missions like collecting debris. We use `ai_001_001` from Hypersim [156] (shown in Fig. 5.a) to model a dense indoor environment. HomeBot mimics the functionality of home-cleaning robots like Roomba [45]. The complexity of such robots is rapidly increasing [160] to enable them to be deployed effectively in intricate indoor settings.

4.4.1 Setting. The environment is dense and dynamic. To have an accurate and up-to-date understanding of the environment, HomeBot captures a series of images by its RGB-D camera, as it moves. These images are then processed to create a 3D model of the scene (*3D scene reconstruction (3DSR)*). HomeBot employs point-based fusion [110, 167, 178] to perform 3DSR. It involves using point clouds, collections of 3D points that represent surfaces and features in a scene, and merging

them together to create a detailed map of the environment. To achieve this, the method needs to *align the point clouds*: it finds the best transformation between two overlapping point clouds and then iteratively refines the alignment until the two clouds are as closely aligned as possible. After that, the point clouds are fused into a single map; HomeBot uses this map to perform its missions.

During the alignment of point clouds, nearest-neighbor search (NNS) is performed to find the closest points in one point cloud to each point in the other point cloud. The goal of this search is to identify the best possible correspondences between points, which are pairs of points that are likely to represent the same physical feature in the real world.



(a) HomeBot Performing Cleaning **(b)** Execution Time

Fig. 5. HomeBot operating in a benchmark environment [156].

The images are 768×1024 pixels, high-quality enough for accurate mapping. We run our experiments on 100 frames of the dataset.

HomeBot employs a reactive planning and control to clean the house, where the cleaning method is determined based on the size of the dirty spot. For small stains, the robot applies a small amount of cleaning solution and uses brushes, while for larger spots, it switches to vacuuming. Once the spot is clean, the robot moves along the free paths to check the rest of the house.

To implement the reactive approach, we use Behavior Tree (BT) [98]. BT is a computational model that is widely used for planning and control in robotics. It is structured as a tree consisting of interconnected nodes, which define conditions (e.g., small or large stain) and actions (e.g., brushing or vacuuming) that must be taken in response to external stimuli.

4.4.2 Evaluation. Fig. 5.b shows HomeBot's compute time on different platforms, where planning and control run on CPUs, and perception runs on GPUs when available. Perception, which performs 3DSR, takes more than 99% of the entire execution time.

3DSR is a hugely data-intensive and highly demanding task. With higher-resolution images, e.g., 12-megapixel iPhone 14 images, which may find applications in fields like Unmanned Little Birds [19], 3DSR will be even more intensive. Recent architecture work [77, 152] suggests hardware acceleration as a promising solution for accelerating 3DSR.

The irregular nature of the algorithm and data structures involved in 3DSR generate *irregular data references*, rendering it a memory-bound task. During reconstruction, the algorithm needs to process the 3D point cloud data, leading to accessing data at irregular intervals based on the point locations being processed. Similarly, in the alignment stage (§4.4.1), the algorithm needs to match and align multiple point clouds from various views, which requires accessing data in non-linear ways—3D points that are *semantically close* to each other in the scene, can be spread *arbitrarily* across the memory layout.

GPUs' high parallelization power make them a superior platform for 3DSR. By partitioning the data and processing each partition simultaneously, 3DSR's operations such as alignment are parallelized, resulting in a significant acceleration—GPUs outperform CPUs by up to 17×.

Moreover, *HG* outperforms *LG* by a factor of 8 due to its larger memory capacity and bandwidth, which can handle the bandwidth-intensive irregular data references generated during 3DSR. On the other hand, *LG* has a weaker memory system that hits the "memory wall" when fully parallelizing the task—the concurrent allocation of large thread-private dynamic data objects and their irregular referencing creates an overwhelming demand for the limited memory capacity and bandwidth of *LG*. §5.2 provides details on this phenomenon.

Finally, BT is fast, taking up to 0.6% of the end-to-end execution, since it selects parameters from a *discrete* set of options instead of a continuous spectrum. This eliminates the need for sophisticated methods to determine optimal parameters, making the algorithm computationally cheap.

Takeaways

• 3DSR is a resource-intensive task that can only be harnessed by HG to achieve real-time performance while being accurate.

• The performance of the memory system is crucial for HomeBot, particularly in its perception stage, which involves many irregular data accesses.

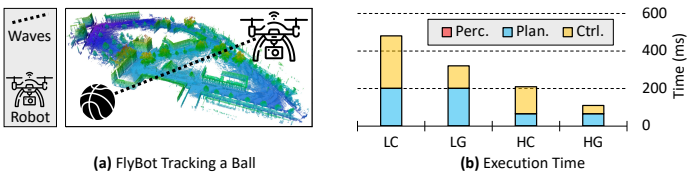
4.4.3 Breadth. HomeBot's computation represents robots that need to precisely understand their surrounding environment and perform subtle tasks. Examples include caregiving robots for elderly and handicapped [179], agriculture robots for crop maintenance [108], and inspection robots for infrastructure upkeep [144].

4.5 FlyBot: Drone Performing Aerial Photography

FlyBot is a pilotless drone that covers sports events on the Freiburg campus [17], tracking a specific object such as a ball and adjusting its position to maintain a seamless view of the target area. Fig. 6.a shows an overview of the environment.

4.5.1 Setting. FlyBot uses a radar sensor to track the ball's position and maintain it in the camera frame in real-time. The sensor emits radio waves that bounce back from the ball to determine its location, based on its radar cross-section, which measures its reflectivity to radar waves. The sensor is calibrated to detect only the ball, as it bounces off the ball and not on other objects, which absorb or scatter the waves.

A lookup table approach, similar to [140], is employed to determine the position of the ball quickly based on sensor data. The table is created by storing readings from the radar sensor for different ball positions offline. During online detection, the table is used to locate the ball. While the table's accuracy may not be high, its performance justifies its use in this application where the exact ball location is not crucial.



(a) FlyBot Tracking a Ball **(b)** Execution Time

Fig. 6. FlyBot operating in a benchmark environment [17].

FlyBot uses the weighted A* (WA*) [154] algorithm to plan paths in the (x, y, z) space. WA* uses a heuristic function to guide the search towards the goal. It creates an environment graph where each node is assigned a cost representing its distance from the starting point. A weight factor, ε, is applied to the heuristic function to balance the exploration and exploitation of the search space: large ε values prioritize faster search, while small ε values prioritize a more optimal path. We use Euclidean norm as the heuristic function and set $\varepsilon = 4$.

FlyBot utilizes model predictive control (MPC) [89] to follow the path identified by the planner. MPC formulates motion control as a *convex optimization problem*, solving it to identify a feasible path that adheres to the robot's constraints. By controlling FlyBot's velocity and acceleration, MPC ensures the robot stays on course while minimizing any deviations from the planned path.

4.5.2 Evaluation. Fig. 6.b shows FlyBot's compute time on different platforms, where the perception and planning stages run on CPUs, while the control stage runs on GPUs when available. Both

planning and control stages contribute significantly to the overall pipeline latency. Planning takes 31%–63%, and control takes 37%–68% of the execution time, depending on the platform. The perception stage involves a computationally cheap table lookup, taking less than 1% of the end-to-end time.

During path planning, the environment graph is searched by generating nodes corresponding to (x, y, z) locations within the environment area and checking whether they are free from collisions. The checking of nodes for collision is inexpensive due to the altitude at which FlyBot operates, which is relatively free from clutter. Therefore, the time taken for path planning is largely determined by the generation of nodes, which is proportional to the size of the environment.

WA* is difficult to parallelize [71], especially when collision detection is inexpensive. The overheads of multi-threading (e.g., locks) may not be justified by the relatively light workloads of nodes. Consequently, single-thread performance is crucial, and this is why high-end computing platforms far outperform low-end ones—the instruction-per-clock (IPC) of *HC* is 2.8, while that of *LC* is 1.2.

The MPC algorithm for generating control commands is expensive. RoBoX [158] proposes a full-fledged hardware for accelerating this process. More than 80% of the control time is spent on solving a constrained vector-valued function through an optimization process. This process involves solving a set of linear equations at each time step, but it can be parallelized and solved through smaller sub-problems. GPUs can take advantage of this parallelism, resulting in high performance.

Notably, MPC benefits from CPU vectorization due to the optimization process involving solving linear equations and performing matrix factorization—both can be readily vectorized. The CPU can perform multiple calculations at once by utilizing SIMD instructions, such as the AVX-512 instructions available in *HC*, resulting in faster computation (see §5.1).

Takeaways

- *FlyBot's performance is bottlenecked by both planning and control stages, with throughput determined by the longest stage. This is influenced by the presence or absence of GPUs in the planning and control platforms.*
- *The planning stage of FlyBot requires high single-threaded performance.*

4.5.3 Breadth. FlyBot's computation represents drones that need fast path planning to perform missions like aerial photography [169], package delivery [135], and traffic monitoring [174].

4.6 CarriBot: Driverless Vehicle Transporting Goods

CarriBot is a driverless vehicle designed to optimize transport processes for time-sensitive materials such as those used in chip manufacturing. In our modeled application, CarriBot navigates Intel's Lab [23] and carries goods, minimizing travel time while avoiding obstacles. Fig. 7.a shows an overview of the application.

4.6.1 Setting. CarriBot operates in a known area: the factory's aerial map is furnished to the robot ahead of transportation. Nevertheless, it must perform real-time mapping to comprehend its immediate vicinity, including the workers and other vehicles in the area, thereby avoiding collisions.

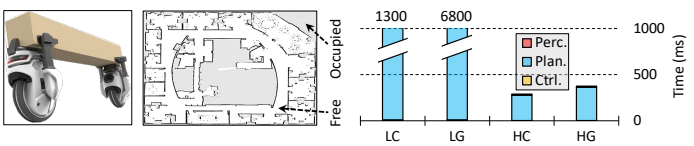


Fig. 7. CarriBot operating in a benchmark environment [23].

A monocular camera is mounted on CarriBot to perform mapping. The camera captures images of the environment, which are then processed to generate "occupancy grids."

Occupancy grids are a way of representing the environment as a grid of cells, where each cell is assigned a value indicating whether it is occupied by an obstacle or not. We use Probabilistic Occupancy Map (POM) [94] to generate occupancy grids from camera images. POM divides the environment into a grid of cells and assigns each cell a probability value that represents the likelihood of it being occupied by an obstacle; cells with probability values above a certain threshold are deemed to be occupied, while those below it are considered to be free.

CarriBot utilizes an Inertial Measurement Unit (IMU) [136] consisting of accelerometers, gyroscopes, and magnetometers. The accelerometers measure linear acceleration, the gyroscopes measure the rate of rotation around axes, and the magnetometers measure the Earth's magnetic field, aiding in determining the vehicle's orientation relative to the magnetic north pole. The EKF algorithm (see §4.2.1) processes the IMU data to localize the vehicle (without mapping).

CarriBot explores the (x, y, θ) space to find the shortest path to the goal. Due to the presence of narrow pathways in the environment and the use of a probabilistic map, CarriBot uses accurate brute-force collision detection to avoid any collision. Different algorithms are used for different platforms: WA* (see §4.5.1) is used for low-end devices, while RA* [62] and GA* [186] are used for *HC* and *HG*, respectively. All the algorithms use the Euclidean distance heuristic with $\varepsilon = 1$.

RA* and GA* improve parallelism in the search process by introducing speculative parallelism on CPUs and proposing a new parallelization method on GPUs, respectively. They improve performance when collision detection is costly (unlike §4.5.2), but are resource-intensive and unsuitable for low-end devices. Notice, they are research state-of-the-art, not established algorithms. We include them to showcase the potential of powerful computing systems to enable sophisticated algorithms for significant speedups in future robotics.

CarriBot uses Dynamic Movement Primitive (DMP) [116, 162] to control its movements along the planned path. DMP is a machine learning technique that decomposes a movement into simpler sub-movements. It uses Gaussian bias functions and shape parameters to *define* the trajectory shape. These parameters are acquired through *imitation learning* [103] and linear regression. Once the parameters are acquired, the final trajectory, including velocity and acceleration, is computed.

4.6.2 Evaluation. Fig. 7.b shows CarriBot's compute time across the platforms, where perception and control run on CPUs, and planning runs on GPUs when available. As noted above, *HC* and *HG* run different planning algorithms.

Processing monocular camera images using POM is computationally cheap, as it relies on a simple probabilistic model instead of expensive 3D reconstruction (see §4.4). Similarly, EKF-based localization is also simple. Unlike EKF-based SLAM (see §4.2), EKF-based localization only estimates the robot's state without including all the landmarks on the map. This significantly reduces the state space (3 variables instead of 15), resulting in a large reduction in compute workload. Overall, perception accounts for less than 1% of the end-to-end execution time.

Path planning is the major component of computation time across all platforms, with collision detection taking up more than 80% of the time. This involves checking many environment locations corresponding to the projected vehicle location to ensure no collision occurs during planning, which is resource-intensive due to the relatively large robot body in the operating environment. RACOD [60] proposes a hardware accelerator for accelerating this process.

RA* and GA* significantly improve search performance by parallelizing intensive collision detection operations on high-end processors, but their resource demands render them unsuitable for low-end devices. RA* lacks the requisite number of cores for speculative parallelization on *LC*, while *LG fails* to run GA* due to its limited memory capacity; the algorithm needs to maintain

thousands of min heaps to find the solution in parallel, which overwhelms the existing memory capacity of *LG* (see §5.2). RA* on *HC* outperforms WA* on *LC* by 4.6×, and GA* on *HG* outperforms WA* on *LG* by 18.2×.

We find CPU vectorization ineffective for collision detection, despite the memory locations checked being nearby and running the same check. This is due to the arbitrary orientations of the robot during planning rendering the layout of memory references axes-unaligned (similar to ray-casting; see §4.1.2), which makes vectorization inapplicable (see §5.1 for details).

Also, the search algorithm's use of irregular data structures like min heaps causes many irregular cache misses, beyond the realm of the evaluated processors' prefetchers (see §5.3). Cache misses are significantly more in GA*, as it employs *thousands* of min heaps, irregularly traversing them.

Finally, DMP's online computation is inexpensive due to its offline learning of motor control primitives, which are stored as weights in a set of basis functions. Online execution involves simply generating new movements using a weighted sum of these basis functions, which is computationally inexpensive: up to 4.1% of the end-to-end execution time.

Takeaways

• Accurate collision detection in planning can be computationally demanding, and low-end platforms may not offer sufficient performance for time-critical applications.

• Algorithmic advancements can lead to the full utilization of hardware potential, resulting in transformative improvements in robotics.

4.6.3 Breadth. CarriBot's computation represents the class of mobile robots that operate in the wild and need to avoid any collision. This includes autonomous vehicles used in transportation such as self-driving cars [153] and delivery robots [163].

5 SYSTEM-LEVEL IMPLICATIONS OF ROBOTICS

Robotics involves diverse workloads, and no single hardware can serve all of them equally well. It is unlikely that vendors will fabricate a specific processor for each robotic application. However, designing processors based on *shared characteristics* across a range of robotic workloads is practical. This section studies the system- and hardware-level implications of robotics, gaining insights into designing "robotics processors."

5.1 Vectorization and Irregular Memory Layouts

All modern CPUs feature vector instructions and large vector registers, such as *LC* and *HC* with 2 KB and 44 KB vector registers respectively. Vectorization is shown to remarkably enhance performance across various workloads [99, 114].

Fig. 8 shows execution time using manual and compiler vectorization, normalized to that without vectorization. We write vectorized code using VCL [4], utilizing input from Intel Advisor [20]. Compiler vectorization is primarily used to enhance the vectorization of third-party codes.

We conclude that, despite the large silicon it occupies, *vectorization does little for most robotic workloads*. FlyBot is an exception to this trend, experiencing 5% speedup with vectorization. FlyBot benefits from vectorization, as it optimizes a vector-valued function in its control stage, as explained in §4.5.2.

Our investigation shows that vectorization is ineffective for operations like ray-casting and collision detection, despite their perceived suitability. The *varying orientation* of the robot with respect to the environment axes leads to axes-unaligned memory accesses (blue arrows in Fig. 2.a),

which current vectorization engines cannot handle.[5] This issue will become more significant for future advanced robots, such as surgical robots that require higher accuracy [59]: operations like collision detection will be more computationally intensive, but current vectorization approaches will not provide a solution.

To address this issue, current vector architectures can incorporate an *additional operand* into vector instructions: a register containing the traversal orientation. This would enable the hardware to compute the memory addresses corresponding to the robot's current layout and vectorize fetching them. Investigating such approaches will be a part of our future work.

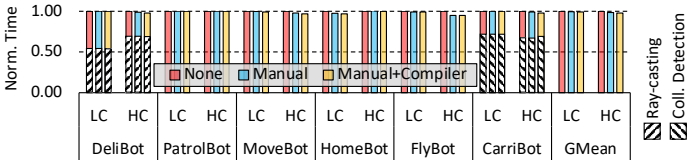


Fig. 8. Vectorization's impact on *LC* and *HC*.

5.2 Parallelism and The Memory Barrier

We find that massive parallelism on the edge, i.e., *LG*, hits the memory wall. The memory problem is twofold: *(i)* limited DRAM capacity and *(ii)* limited DRAM bandwidth.

In detail, we observe that massively parallelizing memory-intensive applications like HomeBot overwhelms the limited DRAM capacity: in-parallel allocation of mega-scale thread-private data (e.g., bookkeeping 3D pixels) dwarfs the available few gigabytes of DRAM. Further, the limited DRAM capacity prevents employing some massively-parallel algorithms (see §4.6.2).

Additionally, on-edge parallelism encounters the memory bandwidth wall, where performance scaling is limited even with increasing cores. Fig. 9 shows the performance of HomeBot, *RoWild*'s most memory-intensive application (on the left side of the graph), and the average of three other memory-intensive applications, namely DeliBot, MoveBot, and FlyBot (on the right side of the graph), with varying the number of threads normalized to single-thread performance. The corresponding bandwidth utilization is shown by yellow dots on the graph.

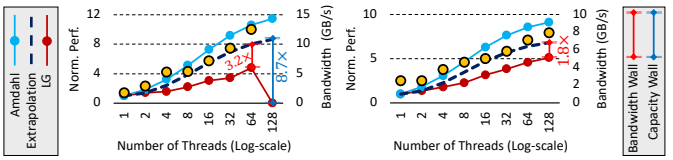


Fig. 9. Performance scaling hits the memory wall. The difference between Amdahl and *LG* is named AG, and the difference between Amdahl and Extrapolation is named EAG.

The complex architecture of GPUs makes it challenging to precisely quantify the impact of memory bandwidth on performance scaling [83, 84]. To estimate this impact, we employ the following methodology: First, we subtract the execution time of the program's serial components from the total execution time to isolate the execution time of the parallel section (t_p). Then, we compute the disparity between the execution time of an n-threaded execution and ideal scaling as defined by Amdahl's law ($\frac{t_p}{n}$); this disparity is termed the *architecture gap (AG)*. Using a linear function, we extrapolate AG from when the bandwidth utilization is low (involving just a few threads) to when it is high (with all threads in use); we name it the *extrapolated AG (EAG)*. Because EAG is derived from scenarios featuring low bandwidth, it is likely *unaffected* by the memory

[5]The `VPMULTISHIFTQB` instruction, introduced in Intel's Cannon Lake, still cannot capture these patterns as the generated memory references exceed the quadword range.

bandwidth wall. Consequently, the contrast between EAG and AG in high-bandwidth settings effectively represents the impact of the memory bandwidth wall.

The results show that *performance scaling on the edge is hindered by the memory wall*. The memory capacity/bandwidth wall alone prevents up to 8.7×/3.2× performance scaling (in HomeBot).

It is worth noting that certain edge platforms have recognized the need for increased memory bandwidth and have substantially improved their offerings [157]. For example, Nvidia's Jetson AGX Orin [26] provides a remarkable 204.8 GB/s of memory bandwidth, which is eight times more than *LG*'s offering. However, this enhancement comes at a considerable cost difference, with AGX Orin's solution priced at $1999 compared to *LG*'s $149, making it cost-prohibitive for many robotic applications. Therefore, we advocate for system-level techniques, such as enhanced data sharing, to improve bandwidth utilization and achieve cost-effective performance scaling. Lastly, high-end platforms like *HC* and *HG* are already overprovisioned in terms of memory bandwidth and do not exhibit this issue.

5.3 Data Prefetching and Complex Access Patterns

LC and *HC* include *simple* stride/stream hardware prefetchers. Also, software prefetching is implemented through compiler optimizations and manual profiler-driven programming (see §3.2). Fig. 10 shows the execution time[6] with prefetching normalized to without it.

The results show disabling hardware prefetchers causes a significant slowdown: *stride prefetching is effective for robotics*. This contrasts with prior works which find stride prefetching to be highly ineffective for various workloads, including commercial [170], cloud [92], and scientific [182].

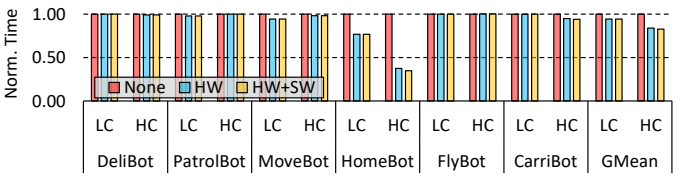


Fig. 10. Prefetching's impact on *LC* and *HC*.

Stride prefetching is effective for two reasons: first, many robotic computations, such as linear algebra, inherently involve strided memory accesses. Second, we deliberately use static data structures, such as arrays, instead of dynamic ones like linked-lists, wherever possible to enable memory accesses to occur in fixed-sized blocks due to the contiguous nature of memory locations. While this approach may reduce the generality of each program's binary and require recompilation for different robots, it is justified by the performance improvement and the application nature, where the robot is aware of its components (e.g., camera) and can specialize software based on them.

Nevertheless, *simple prefetchers are inadequate for memory-intensive robotic applications*, as they do not capture complex memory patterns generated by the employed irregular data structures like min heaps and hashmaps. This results in a large post-prefetch LLC misses per kilo instruction (MPKI), with CarriBot having an MPKI of 1.5 (2.1) and HomeBot having an MPKI of 3.5 (1.4) on *LC* (*HC*, respectively).

Finally, software prefetching offers little improvements. In DeliBot, PatrolBot, FlyBot, and CarriBot, there is insignificant opportunity for software prefetching. In MoveBot and HomeBot, there is some opportunity, but prefetches are not timely–they hide the latency of one or a few loop

[6]Hit ratio is not a good indicator of prefetching effectiveness in *HC*, as different prefetchers are for *different* cache levels: `Hardware Prefetcher` (L2), `Adjacent Cache Prefetch` (L1), `DCU Streamer Prefetcher` (L1), `DCU IP Prefetcher` (L1), and `LLC Prefetcher` (L3).

iterations. In HomeBot, additionally, excessive software prefetching can degrade performance due to a full L1D fill buffer of pending misses [100].

5.4 Caches and Data Movements

Prior work [69, 123, 130, 165] reports "data movement" as the primary cause of slowdown and inefficiency in processors. Fig. 11 shows the percentage of used bytes out of all fetched bytes for every program. We get the trace of memory accesses using Intel Pin 3.25 [39] and process them offline. We consider 64 B cachelines, which is the configuration of the CPUs.

The results show that *caches perform excessive unnecessary data movements (UDMs)*. In HomeBot, only 33% of the fetched data is used; and, on average across all, only 56% of the fetched data is used (cf. 76%, 92%, 80% average utilization reported for SQL [80], scientific [79], SPEC [111] workloads, respectively). This shows how caches are overprovisioned and are unaware of robotic semantics.

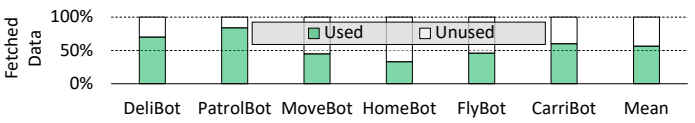


Fig. 11. Data movement.

Caches are architected partly based on the "traditional spatial locality" concept; they use large blocks to capture *(i)* large data types (e.g., 8-byte `double`), and *(ii)* nearby data items (e.g., 1D arrays). However, memory accesses are not always so in robotics. For example, in ray-casting and collision detection—two dominant kernels—memory is traversed mostly in an *oriented* manner (see §5.1), and every memory request accesses only *one bit*; caches are in fact working *against* application semantics, resulting in significant UDMs.

Notice, in this experiment, we do not model *(i)* hardware prefetches (Pin traces lack) and *(ii)* cache size (infinite-size cache is modeled). Hence, 44% useless fetches we report is an *underestimate*. Also, with larger cachelines (e.g., in GPUs), UDMs increase; e.g., by 1.22× with 128 B lines.

To address this, caches need awareness of the robot's body and traversal. This could be achieved by extending classic approaches like [119] that fetch likely-to-be-used parts of cachelines or [86] that dynamically adjust cache block sizes. Either way, the semantics must be transferred to hardware.

6 DISCUSSION

In this section, we delve into two important questions in robotic benchmarking, highlighting the performance specifications future computing platforms must meet in order to facilitate the widespread rollout of real-time robotics.

6.1 What Performance Is Needed for "Real-Time" Robotics?

The designation of "real-time" performance in the multi-staged software pipeline of robotics hinges on its specific application. Various studies highlight differing latency benchmarks: for example, the perception stages have reported constraints from 30 ms [90] to 200 ms [76], while the planning stages have ranged from sub-millisecond [142] to 250 ms [70]. Given these disparities, it is imperative to tailor the architecture of the robotic processor for each stage, ensuring it meets the real-time demands of diverse applications.

Moreover, robotic tasks often present a trade-off between accuracy and execution duration. Greater accuracy demands more computational resources, leading to extended execution times. Through targeted optimization of platforms for specific tasks, it is possible to achieve real-time performance without significantly sacrificing accuracy.

Real-time performance is also a function of the speed at which the robot moves (and hence must react to changes in its environment): autonomous vehicles traveling at highway speeds have stricter

performance requirements than legged robots moving at walking pace. Furthermore, the more complex and dynamic the environment, the greater the challenge to achieve real-time performance. Thus, the need for accelerating robot computation will continue unabated into the foreseeable future.

6.2 Ranking The Computers: Which Platform Should I Buy for My Robot?

As illustrated in §4, the computational needs of diverse robotic tasks and applications vary substantially, indicating that certain platforms may be better suited for specific contexts. Further, as detailed in §3.3, these platforms differ in cost and TDP (among other key features), prompting deliberation about the most effective computational platform for robotics.

In other domains, existing research suggests quantitative criteria for evaluating computational platforms, with metrics like performance per watt and performance per unit area [131] being notably recognized. However, in robotics, these metrics often fall short in encapsulating the actual demands. For instance, in robotics, the power allocated to computing, unlike in, say, data centers, forms a minor portion of the overall power usage (§3.5). Similarly, the metric of performance per area might not be fully representative. Even when considering metrics like performance relative to the total cost of ownership (TCO) [101], it is essential to note that a robot's price may be heavily influenced by non-computational components. To illustrate, a self-driving car priced at over $100,000 might easily incorporate processors costing thousands of dollars, whereas a budget-constrained wildlife conservation robot might not.

Determining the best computational platforms for robotics is a complex endeavor, especially as real-time functionality is not just pivotal for user experience but also for safety. For instance, the prompt detection of obstacles and timely response can be lifesaving in autonomous vehicles. Continued research in robotic-architecture will introduce quantitative criteria for designing efficient robotic processors. This will involve the consideration of elements like safety, raw performance, cost per user ownership, projected revenue, and more.

7 CONCLUSION

In this paper, we introduced *RoWild*, a comprehensive benchmark suite for robotics, and used it to conduct a systematic study of robotics on modern hardware. With this suite and this study, we aim to drive research at the intersection of robotics and systems to improve the performance of robotics.

The realm of robotics is in a state of constant transformation, producing myriad research publications and industrial innovations annually. One cannot hope to encompass the entirety of the field in a single study and static benchmark suite. Hence, our future work will continuously augment *RoWild* with additional algorithms, applications, and analyses as the field progresses, and we welcome contributions by the open-source community.

Our results show that many robotic applications struggle to achieve real-time performance, especially when running on low-end platforms. As a result, current robots often have to sacrifice accuracy to meet real-time requirements. For instance, EureCar URV, a self-driving car, uses an *imprecise* collision detection to maintain acceptable execution time [122]. However, this trade-off will not be sustainable for future robots with high responsiveness and accuracy demands. Moreover, in the coming years, robotic computation is set to become more intense and complex due to the increasing demand for robots to perform sophisticated tasks, particularly in critical applications such as medical procedures or in crowded public settings. Achieving the required level of responsiveness and accuracy will necessitate advanced algorithms and more intense computation.

All this underscores the pressing need to identify and address architectural inefficiencies—vectorization, prefetching, data movement, and many more to come—in order to achieve real-time performance in the robots of tomorrow.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grant CCF-2028949, by a VMware University Research Fund Award, and by the Parallel Data Lab (PDL) Consortium (Alibaba, Amazon, Datrium, Facebook, Google, Hewlett-Packard Enterprise, Hitachi, IBM, Intel, Microsoft, NetApp, Oracle, Salesforce, Samsung, Seagate, and TwoSigma). Mohammad Bakhshalipour was supported by the Apple CMU ECE Ph.D. Fellowship in Integrated Systems. We would like to thank the anonymous reviewers and our shepherd, Y.C. Tay, for their valuable feedback.

REFERENCES

[1] [n. d.]. Search-Based Planning Lab. http://www.sbpl.net/.
[2] 2012. LoCoBot: An Open Source Low Cost Robot. http://www.locobot.org/.
[3] 2019. How Robots Change the World. https://resources.oxfordeconomics.com/how-robots-change-the-world/.
[4] 2022. C++ Vector Class Library Version 2. https://www.agner.org/optimize/vcl_manual.pdf.
[5] n.d.. 3DR Solo Drone. https://uavsystemsinternational.com/products/3dr-solo-drone.
[6] n.d.. AscTec Pelican. https://www.aeroexpo.online/prod/ascending-technologies/product-181442-24426.html.
[7] n.d.. Boston Dynamics' Atlas. https://www.bostondynamics.com/atlas.
[8] n.d.. Boston Dynamics' Spot Robot. https://www.bostondynamics.com/products/spot.
[9] n.d.. Boxbot Launches Last-Mile, Self-Driving Parcel Delivery System. https://www.roboticsbusinessreview.com/supply-chain/boxbot-launches-last-mile-self-driving-parcel-delivery-system/.
[10] n.d.. Bullet Collision Detection & Physics Library. https://pybullet.org/Bullet/BulletFull/.
[11] n.d.. Carnegie Mellon University, Wean Hall (WEH). https://www.cmu.edu/computing/services/teach-learn/tes/classrooms/locations/wean.html.
[12] n.d.. Clearpath Robotics' Grizzly Robot. https://clearpathrobotics.com/blog/tag/grizzly/.
[13] n.d.. CppRobotics. https://github.com/onlytailei/CppRobotics/.
[14] n.d.. EVGA GeForce GTX TITAN X Superclocked Graphics Card. EVGAGeForceGTXTITANXSuperclockedGraphicsCard.
[15] n.d.. FANUC's M-2000iA/2300 Robot. https://www.fanucamerica.com/products/robots/series/m-2000ia/m-2000ia-2300-heavy-payload-robot.
[16] n.d.. Franka Emika's Panda Robot. https://www.pomorobotics.com/robots/frankpanda/.
[17] n.d.. Freiburg Campus 360 Degree 3D Scans. http://ais.informatik.uni-freiburg.de/projects/datasets/fr360/.
[18] n.d.. HUSKY UGV. https://clearpathrobotics.com/assets/guides/foxy/husky/index.html.
[19] n.d.. In a First, Full-Sized Robo-Copter Flies With No Human Help. https://www.wired.com/2010/07/in-a-first-full-sized-robo-copter-flies-with-no-human-help/.
[20] n.d.. Intel Advisor: Design Code for Efficient Vectorization, Threading, Memory Usage, and Accelerator Offloading. https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html.
[21] n.d.. Intel Movidius Vision Processing Units. https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html.
[22] n.d.. Intel RealSense Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html.
[23] n.d.. Intel Reserach Lab; The Raw Log Data Was Provided by Dirk Haehnel. http://www2.informatik.uni-freiburg.de/%7Estachnis/datasets/datasets/intel-lab/intel.gfs.png.
[24] n.d.. Intel Xeon Gold 5218R Processor. https://ark.intel.com/content/www/us/en/ark/products/199342/intel-xeon-gold-5218r-processor-27-5m-cache-2-10-ghz.html.
[25] n.d.. JACKAL UNMANNED GROUND VEHICLE. https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/.
[26] n.d.. Jetson AGX Orin Series. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/.
[27] n.d.. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.
[28] n.d.. KUKA. https://www.kuka.com/en-us.
[29] n.d.. KUKA KR 60-3 Robot. https://robodk.com/robot/KUKA/KR-60-3.
[30] n.d.. LBR Iiwa. https://www.kuka.com/en-us/products/robotics-systems/industrial-robots/lbr-iiwa.

[31] n.d.. Maximize the Efficiency of Your Logistics Operations with Robots from MiR. https://www.mobile-industrial-robots.com/.
[32] n.d.. NVIDIA Clara. https://docs.nvidia.com/clara/index.html.
[33] n.d.. NVIDIA DRIVE End-To-End Solutions for Autonomous Vehicles. https://developer.nvidia.com/drive.
[34] n.d.. NVIDIA Isaac: The Accelerated Platform for Robotics and AI. https://www.nvidia.com/en-us/deep-learning-ai/industries/robotics/.
[35] n.d.. NVIDIA Jetson: Accelerating Next-Gen Edge AI and Robotics. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.
[36] n.d.. NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems.
[37] n.d.. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.
[38] n.d.. Phantom 4 Pro - DJI. https://www.dji.com/phantom-4-pro.
[39] n.d.. Pin 3.25 Release Notes. https://software.intel.com/sites/landingpage/pintool/docs/98650/README.
[40] n.d.. Pioneer 3-DX. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf.
[41] n.d.. PR2. https://www.wevolver.com/specs/pr2.
[42] n.d.. PythonRobotics. https://github.com/AtsushiSakai/PythonRobotics/.
[43] n.d.. Robotics Automation for Warehousing, 3PLs, Distribution, Manufacturing. https://fetchrobotics.com/.
[44] n.d.. Roomba 980 Robot Vacuum. https://www.irobot.com/en_US/roomba-vacuuming-robot-irobot-roomba-restored-980/R980R99.html.
[45] n.d.. Roomba I7+ Self-Emptying Robot Vacuum. https://www.irobot.com/en_US/roomba-vacuuming/robot-vacuum-irobot-roomba-i7-plus/I755020.html.
[46] n.d.. ROS - Robot Operating System. https://www.ros.org/.
[47] n.d.. Skydio. https://www.skydio.com/.
[48] n.d.. SoftBank Robotics' Pepper Robot. https://us.softbankrobotics.com/pepper.
[49] n.d.. TALOS: The Walking Humanoid Robot That Integrates the Latest Cutting-Edge Robotics Technology. https://pal-robotics.com/robots/talos/.
[50] n.d.. The Open Motion Planning Library. http://ompl.kavrakilab.org/.
[51] n.d.. The Parallella Board. https://parallella.org.
[52] n.d.. The UR10e. https://www.universal-robots.com/products/ur10-robot/.
[53] n.d.. The UR5e. https://www.universal-robots.com/products/ur5-robot/.
[54] n.d.. TIAGo. https://pal-robotics.com/robots/tiago/.
[55] n.d.. TurtleBot. https://www.turtlebot.com/.
[56] n.d.. TurtleBot3. https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.
[57] n.d.. Xilinx Adaptive SoCs. https://www.xilinx.com/products/silicon-devices/soc.html.
[58] n.d.. YuMi - IRB 14000 | Collaborative Robot. https://new.abb.com/products/robotics/robots/collaborative-robots/yumi/irb-14000-yumi.
[59] Aleks Attanasio, Bruno Scaglioni, Elena De Momi, Paolo Fiorini, and Pietro Valdastri. 2021. Autonomy in Surgical Robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), 651–679. https://doi.org/10.1146/annurev-control-062420-090543
[60] Mohammad Bakhshalipour, Seyed Borna Ehsani, Mohamad Qadri, Dominic Guri, Maxim Likhachev, and Phillip B Gibbons. 2022. RACOD: Algorithm/Hardware Co-Design for Mobile Robot Path Planning. In *Int'l Symp. in Computer Architecture (ISCA)*. IEEE/ACM. https://doi.org/10.1145/3470496.3527383
[61] Mohammad Bakhshalipour, Maxim Likhachev, and Phillip B. Gibbons. 2022. RTRBench: A Benchmark Suite for Real-Time Robotics. In *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS55109.2022.00024 https://cmu-roboarch.github.io/rtrbench/.
[62] Mohammad Bakhshalipour, Mohamad Qadri, Dominic Guri, Seyed Borna Ehsani, Maxim Likhachev, and Phillip Gibbons. 2023. Runahead A*: Speculative Parallelism for A* with Slow Expansions. In *Int'l Conf. on Automated Planning and Scheduling (ICAPS)*.
[63] Gelareh Bakhtyar, Vi Nguyen, Mecit Cetin, and Duc Nguyen. 2016. Backward Dijkstra Algorithms for Finding the Departure Time Based on the Specified Arrival Time for Real-Life Time-Dependent Networks. *Journal of Applied Mathematics and Physics* 4, 1 (2016). https://doi.org/10.4236/jamp.2016.41001
[64] Calin Belta, Antonio Bicchi, Magnus Egerstedt, Emilio Frazzoli, Eric Klavins, and George J Pappas. 2007. Symbolic Planning and Control of Robot Motion [grand Challenges of Robotics]. *IEEE Robotics & Automation Magazine* 14, 1 (2007), 61–70. https://doi.org/10.1109/MRA.2007.339624
[65] Joshua Bialkowski, Sertac Karaman, and Emilio Frazzoli. 2011. Massively Parallelizing the RRT and the RRT*. In *Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 3513–3518. https://doi.org/10.1109/IROS.2011.6095053
[66] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Wenzhi Cui, Aleksandra Faust, and Vijay Reddi. 2018. MAVBench: Micro Aerial Vehicle Benchmarking. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 894–907.

https://doi.org/10.1109/MICRO.2018.00077

[67] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Bardienus Pieter Duisterhof, Brian Plancher, Kayvan Mansoorshahi, Marcelino Almeida, Wenzhi Cui, Aleksandra Faust, and Vijay Janapa Reddi. 2022. The Role of Compute in Autonomous Micro Aerial Vehicles: Optimizing for Mission Time and Energy Efficiency. *ACM Trans. Comput. Syst.* 39, 1–4, Article 3 (jul 2022), 44 pages. https://doi.org/10.1145/3511210

[68] Behzad Boroujerdian, Ying Jing, Devashree Tripathy, Amit Kumar, Lavanya Subramanian, Luke Yen, Vincent Lee, Vivek Venkatesan, Amit Jindal, Robert Shearer, et al. 2023. FARSI: An Early-Stage Design Space Exploration Framework to Tame the Domain-Specific System-On-Chip Complexity. *ACM Transactions on Embedded Computing Systems (TECS)* 22, 2 (2023), 1–35. https://doi.org/10.1145/3544016

[69] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 316–331. https://doi.org/10.1145/3173162.3173177

[70] Chris Bowen and Ron Alterovitz. 2014. Closed-Loop Global Motion Planning for Reactive Execution of Learned Tasks. In *Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 1754–1760. https://doi.org/10.1109/IROS.2014.6942792

[71] Sandy Brand and Rafael Bidarra. 2012. Multi-Core Scalable and Efficient Pathfinding with Parallel Ripple Search. *computer animation and virtual worlds* 23, 2 (2012), 73–85. https://doi.org/10.1007/978-3-642-25090-3_25

[72] Mihai Bujanca, Paul Gafton, Sajad Saeedi, Andy Nisbet, Bruno Bodin, Michael FP O'Boyle, Andrew J Davison, Paul HJ Kelly, Graham Riley, Barry Lennox, Mikel Luján, and Steve Furber. 2019. SLAMBench 3.0: Systematic Automated Reproducible Evaluation of SLAM Systems for Robot Vision Challenges and Scene Understanding. In *Int'l Conf. on Robotics and Automation (ICRA)*. IEEE, 6351–6358. https://doi.org/10.1109/ICRA.2019.8794369

[73] Wolfram Burgard, Cyrill Stachniss, Giorgio Grisetti, Bastian Steder, Rainer Kümmerle, Christian Dornhege, Michael Ruhnke, Alexander Kleiner, and Juan D Tardös. 2009. A Comparison of SLAM Algorithms Based on a Graph of Relations. In *Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2089–2095. https://doi.org/10.1109/IROS.2009.5354691

[74] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics* 37, 6 (2021), 1874–1890. https://doi.org/10.1109/TRO.2021.3075644

[75] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation. In *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)* (Seattle, Washington) *(SC '11)*. ACM, NY, Article 52, 12 pages. https://doi.org/10.1145/2063384.2063454

[76] Aayush K Chaudhary, Rakshit Kothari, Manoj Acharya, Shusil Dangi, Nitinraj Nair, Reynold Bailey, Christopher Kanan, Gabriel Diaz, and Jeff B Pelz. 2019. RITnet: Real-Time Semantic Segmentation of The Eye for Gaze Tracking. In *Int'l Conf. on Computer Vision Workshop (ICCVW)*. IEEE, 3698–3702. https://doi.org/10.1109/ICCVW.2019.00568

[77] Faquan Chen, Rendong Ying, Jianwei Xue, Fei Wen, and Peilin Liu. 2023. ParallelNN: A Parallel Octree-Based Nearest Neighbor Search Accelerator for 3D Point Clouds. In *Int'l Symp. on High-Performance Computer Architecture (HPCA)*. IEEE, 403–414. https://doi.org/10.1109/HPCA56546.2023.10070940

[78] Jainwei Chen, Lakshmi Kumar Dabbiru, Daniel Wong, Murali Annavaram, and Michel Dubois. 2010. Adaptive and Speculative Slack Simulations of CMPs on CMPs. In *Int'l Symp. on Microarchitecture (MICRO) (MICRO '43)*. IEEE Computer Society, 523–534. https://doi.org/10.1109/MICRO.2010.47

[79] Yen-Hao Chen and Yi-Yu Liu. 2013. Dual-Addressing Memory Architecture for Two-Dimensional Memory Access Patterns. In *2013 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 71–76. https://doi.org/10.7873/DATE.2013.029

[80] Trishul M Chilimbi, Bob Davidson, and James R Larus. 1999. Cache-Conscious Structure Definition. In *Proceedings of the ACM SIGPLAN 1999 conference on Programming Language Design and Implementation (PLDI)*. 13–24. https://doi.org/10.1145/301618.301635

[81] Chieh Chung and Chia-Hsiang Yang. 2019. A Distributed Autonomous and Collaborative Multi-Robot System Featuring a Low-Power Robot SoC in 22nm CMOS for Integrated Battery-Powered Minibots. In *Int'l Solid-State Circuits Conf. (ISSCC)*. IEEE, 48–50. https://doi.org/10.1109/ISSCC.2019.8662463

[82] Chieh Chung and Chia-Hsiang Yang. 2020. A 1.5-μJ/Task Path-Planning Processor for 2-D/3-D Autonomous Navigation of Microrobots. *IEEE Journal of Solid-State Circuits (JSSC)* 56, 1 (2020), 112–122. https://doi.org/10.1109/JSSC.2020.3037138

[83] Sina Darabi, Mohammad Sadrosadati, Negar Akbarzadeh, Joël Lindegger, Mohammad Hosseini, Jisung Park, Juan Gómez-Luna, Onur Mutlu, and Hamid Sarbazi-Azad. 2022. Morpheus: Extending the Last Level Cache Capacity in GPU Systems Using Idle GPU Core Resources. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 228–244.

[84] Sina Darabi, Ehsan Yousefzadeh-Asl-Miandoab, Negar Akbarzadeh, Hajar Falahati, Pejman Lotfi-Kamran, Mohammad Sadrosadati, and Hamid Sarbazi-Azad. 2022. OSM: Off-Chip Shared Memory for GPUs. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 33, 12 (2022), 3415–3429. https://doi.org/10.1109/TPDS.2022.3154315

[85] Jeffrey Delmerico and Davide Scaramuzza. 2018. A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots. In *IEEE international conference on robotics and automation (ICRA)*. IEEE, 2502–2509. https://doi.org/10.1109/ICRA.2018.8460664

[86] Czarek Dubnicki and Thomas J LeBlanc. 1992. Adjustable Block Size Coherent Caches. In *Int'l Symp. in Computer Architecture (ISCA)*. 170–180. https://doi.org/10.1109/ISCA.1992.753314

[87] Sankeerth Durvasula, Raymond Kiguru, Samarth Mathur, Jenny Xu, Jimmy Lin, and Nandita Vijaykumar. 2022. VoxelCache: Accelerating Online Mapping in Robotics and 3D Reconstruction Tasks. *arXiv preprint arXiv:2210.08729* (2022). https://doi.org/10.1145/3559009.3569675

[88] Christer Ericson. 2004. *Real-Time Collision Detection*. CRC Press. https://doi.org/10.1201/b14581

[89] Farbod Farshidian, Edo Jelavic, Asutosh Satapathy, Markus Giftthaler, and Jonas Buchli. 2017. Real-Time Motion Planning of Legged Robots: A Model Predictive Control Approach. In *IEEE-RAS Int'l Conf. on Humanoid Robotics (Humanoids)*. IEEE, 577–584. https://doi.org/10.1109/HUMANOIDS.2017.8246930

[90] Yu Feng, Nathan Goulding-Hotta, Asif Khan, Hans Reyserhove, and Yuhao Zhu. 2022. Real-Time Gaze Tracking with Event-Driven Eye Segmentation. In *IEEE Conf. on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 399–408. https://doi.org/10.1109/VR51125.2022.00059

[91] Yu Feng, Boyuan Tian, Tiancheng Xu, Paul Whatmough, and Yuhao Zhu. 2020. Mesorasi: Architecture Support for Point Cloud Analytics Via Delayed-Aggregation. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 1037–1050. https://doi.org/10.1109/MICRO50266.2020.00087

[92] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (London, England, UK). ACM, NY, 37–48. https://doi.org/10.1145/2150976.2150982

[93] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. 2006. Replanning with RRTs. In *Int'l Conf. on Robotics and Automation (ICRA)*. IEEE, 1243–1248. https://doi.org/10.1109/ROBOT.2006.1641879

[94] Chaitanyavishnu S Gadde, Mohitvishnu S Gadde, Nishant Mohanty, and Suresh Sundaram. 2021. Fast Obstacle Avoidance Motion in Small Quadcopter Operation in a Cluttered Environment. In *2021 IEEE Int'l Conf. on Electronics, Computing and Communication Technologies (CONECCT)*. IEEE, 1–6. https://doi.org/10.1109/CONECCT52877.2021.9622631

[95] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. 2014. Informed RRT*: Optimal Sampling-Based Path Planning Focused Via Direct Sampling of an Admissible Ellipsoidal Heuristic. In *Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2997–3004. https://doi.org/10.1109/IROS.2014.6942976

[96] Yiming Gan, Yu Bo, Boyuan Tian, Leimeng Xu, Wei Hu, Shaoshan Liu, Qiang Liu, Yanjun Zhang, Jie Tang, and Yuhao Zhu. 2021. Eudoxus: Characterizing and Accelerating Localization in Autonomous Machines Industry Track Paper. In *Int'l Symp. on High-Performance Computer Architecture (HPCA)*. IEEE, 827–840. https://doi.org/10.1109/HPCA51647.2021.00074

[97] Roland Geraerts and Mark H Overmars. 2007. Creating High-Quality Paths for Motion Planning. *The international journal of robotics research* 26, 8 (2007), 845–863. https://doi.org/10.1177/0278364907079280

[98] Razan Ghzouli, Swaib Dragule, Thorsten Berger, Einar Broch Johnsen, and Andrzej Wasowski. 2022. Behavior Trees and State Machines in Robotics Applications. *arXiv preprint arXiv:2208.04211* (2022).

[99] Constantino Gómez, Filippo Mantovani, Erich Focht, and Marc Casas. 2021. Efficiently Running SpMV on Long Vector Architectures. In *Proceedings of the 26th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP)*. 292–303. https://doi.org/10.1145/3437801.3441592

[100] Zhangxiaowen Gong, Houxiang Ji, Yao Yao, Christopher W. Fletcher, Christopher J. Hughes, and Josep Torrellas. 2022. Graphite: Optimizing Graph Neural Networks on CPUs Through Cooperative Software-Hardware Techniques. In *Proceedings of the 49th Annual Int'l Symp. on Computer Architecture* (NY) *(ISCA '22)*. ACM, NY, 916–931. https://doi.org/10.1145/3470496.3527403

[101] Boris Grot, Damien Hardy, Pejman Lotfi-Kamran, Babak Falsafi, Chrysostomos Nicopoulos, and Yiannakis Sazeides. 2012. Optimizing Data-Center TCO with Scale-Out Processors. *IEEE Micro* 32, 5 (2012), 52–63. https://doi.org/10.1109/MM.2012.71

[102] Alejandro Hidalgo-Paniagua, Juan Pedro Bandera, Manuel Ruiz-de Quintanilla, and Antonio Bandera. 2018. Quad-RRT: A Real-Time GPU-Based Global Path Planner in Large-Scale Real Environments. *Expert Systems with Applications* 99 (2018), 141–154. https://doi.org/10.1016/j.eswa.2018.01.035

[103] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation Learning: A Survey of Learning Methods. *Comput. Surveys* 50, 2 (2017), 1–35. https://doi.org/10.1145/3054912

[104] Aamer Jaleel, Robert S Cohn, Chi-Keung Luk, and Bruce Jacob. 2008. CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator. In *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*. 28–36.

[105] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. 2020. RLBench: The Robot Learning Benchmark & Learning Environment. *IEEE Robotics and Automation Letters* 5, 2 (2020), 3019–3026. https://doi.org/10.1109/LRA.2020.2974707

[106] Noémie Jaquier, Leonel Rozo, Sylvain Calinon, and Mathias Bürger. 2020. Bayesian Optimization Meets Riemannian Manifolds in Robot Learning. In *Conf. on Robot Learning*. PMLR, 233–246. https://doi.org/10.48550/arXiv.1910.04998

[107] Katharin R Jensen-Nau, Tucker Hermans, and Kam K Leang. 2020. Near-Optimal Area-Coverage Path Planning of Energy-Constrained Aerial Robots with Application in Autonomous Environmental Monitoring. *IEEE Transactions on Automation Science and Engineering* 18, 3 (2020), 1453–1468.

[108] Ailian Jiang and Tofael Ahamed. 2023. Navigation of an Autonomous Spraying Robot for Orchard Operations Using LiDAR for Tree Trunk Detection. *Sensors* 23, 10 (2023), 4808.

[109] Monodeep Kar, Amit Agarwal, Steven Hsu, David Moloney, Gregory Chen, Raghavan Kumar, Huseyin Sumbul, Phil Knag, Mark Anders, Himanshu Kaul, Jonathan Byrne, Luca Sarti, Ram Krishnamurthy, and Vivek De. 2020. A Ray-Casting Accelerator in 10nm CMOS for Efficient 3D Scene Reconstruction in Edge Robotics and Augmented Reality Applications. In *IEEE Symp. on VLSI Circuits (VLSIC)*. IEEE, 1–2. https://doi.org/10.1109/VLSICircuits18222.2020.9163067

[110] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. 2013. Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion. In *Int'l Conf. on 3D Vision (3DV)*. IEEE, 1–8. https://doi.org/10.1109/3DV.2013.9

[111] Georgios Keramidas, Michail Mavropoulos, Anna Karvouniari, and Dimitris Nikolos. [n. d.]. Instruction Based Management of Faulty Data Caches. http://students.ceid.upatras.gr/~mavropoulo/abstract_acaces.pdf. ([n. d.]).

[112] Andrew Kerr, Gregory Diamos, and Sudhakar Yalamanchili. 2009. *A Characterization and Analysis of GPGPU Kernels*. Technical Report. Georgia Institute of Technology.

[113] Youchang Kim, Dongjoo Shin, Jinsu Lee, Yongsu Lee, and Hoi-Jun Yoo. 2016. A 0.55V 1.1mW Artificial-Intelligence Processor with PVT Compensation for Micro Robots. In *Int'l Solid-State Circuits Conf. (ISSCC)*. IEEE, 258–259. https://doi.org/10.1109/ISSCC.2016.7418005

[114] Kazuhiko Komatsu, Shintaro Momose, Yoko Isobe, Osamu Watanabe, Akihiro Musa, Mitsuo Yokokawa, Toshikazu Aoyama, Masayuki Sato, and Hiroaki Kobayashi. 2018. Performance Evaluation of a Vector Supercomputer SX-Aurora TSUBASA. In *Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 685–696. https://doi.org/10.1109/SC.2018.00057

[115] Richard E Korf. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial intelligence* 27, 1 (1985), 97–109. https://doi.org/10.1016/0004-3702(85)90084-0

[116] Leonidas Koutras and Zoe Doulgeri. 2020. Dynamic Movement Primitives for Moving Goals with Temporal Scaling Adaptation. In *Int'l Conf. on Robotics and Automation (ICRA)*. IEEE, 144–150. https://doi.org/10.1109/ICRA40945.2020.9196765

[117] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Ninad Jadhav, Aleksandra Faust, and Vijay Janapa Reddi. 2022. Roofline Model for UAVs: A Bottleneck Analysis Tool for Onboard Compute Characterization of Autonomous Unmanned Aerial Vehicles. In *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS55109.2022.00023

[118] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Paul Whatmough, Aleksandra Faust, Sabrina Neuman, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. 2022. Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 300–317. https://doi.org/10.1109/MICRO56248.2022.00033

[119] Sanjeev Kumar and Christopher Wilkerson. 1998. Exploiting Spatial Locality in Data Caches Using Spatial Footprints. In *Int'l Symp. in Computer Architecture (ISCA)*. IEEE, 357–368. https://doi.org/10.1109/ISCA.1998.694794

[120] Steven M LaValle. 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning. (1998).

[121] Steven M LaValle and James J Kuffner Jr. 2001. Randomized Kinodynamic Planning. *The international journal of robotics research* 20, 5 (2001), 378–400. https://doi.org/10.1109/ROBOT.1999.770022

[122] Unghui Lee, Jiwon Jung, Seunghak Shin, Yongseop Jeong, Kibaek Park, David Hyunchul Shim, and In-so Kweon. 2016. EureCar Turbo: A Self-Driving Car That Can Handle Adverse Weather Conditions. In *Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2301–2306. https://doi.org/10.1109/IROS.2016.7759359

[123] Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. 2020. There's Plenty of Room at the Top: What Will Drive Computer Performance After Moore's Law? *Science* 368, 6495 (2020), eaam9744.

[124] John J Leonard, David A Mindell, and Erik L Stayton. 2020. Autonomous Vehicles, Mobility, and Employment Policy: The Roads Ahead. *Massachusetts Institute of Technology, Cambridge, MA, Rep. RB02-2020* (2020).

[125] Qingqing Li, Jorge Peña Queralta, Tuan Nguyen Gia, Zhuo Zou, and Tomi Westerlund. 2020. Multi-Sensor Fusion for Navigation and Mapping in Autonomous Vehicles: Accurate Localization in Urban Environments. *Unmanned Systems*

8, 03 (2020), 229–237.
[126] Shiqi Lian, Yinhe Han, Xiaoming Chen, Ying Wang, and Hang Xiao. 2018. Dadu-P: A Scalable Accelerator for Robot Motion Planning in a Dynamic Environment. In *Design Automation Conf. (DAC)*. IEEE, 1–6.
[127] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The Architectural Implications of Autonomous Driving: Constraints and Acceleration. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 751–766. https://doi.org/10.1145/3173162.3173191
[128] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *European Conf. on Computer Vision (ECCV)*. Springer, 740–755. https://doi.org/10.48550/arXiv.1405.0312
[129] Weizhuang Liu, Bo Yu, Yiming Gan, Qiang Liu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. 2021. Archytas: A Framework for Synthesizing and Dynamically Optimizing Accelerators for Robotic Localization. In *Int'l Symp. on Microarchitecture (MICRO)*. 479–493. https://doi.org/10.1145/3466752.3480077
[130] Elliot Lockerman, Axel Feldmann, Mohammad Bakhshalipour, Alexandru Stanescu, Shashwat Gupta, Daniel Sanchez, and Nathan Beckmann. 2020. Livia: Data-Centric Computing Throughout the Memory Hierarchy. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 417–433. https://doi.org/10.1145/3373376.3378497
[131] Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, and Babak Falsafi. 2012. Scale-Out Processors. In *Int'l Symp. in Computer Architecture (ISCA)*. 500–511. https://doi.org/10.1145/2366231.2337217
[132] A Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra. 2018. Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *Conf. on Robot Learning*. PMLR, 561–591.
[133] AV Malakhov, DV Shutin, and SG Popov. 2020. Bricklaying Robot Moving Algorithms at a Construction Site. In *IOP Conf. Series: Materials Science and Engineering*, Vol. 734. IOP Publishing, 012126.
[134] Víctor Mayoral-Vilches, Sabrina M Neuman, Brian Plancher, and Vijay Janapa Reddi. 2022. Robotcore: An Open Architecture for Hardware Acceleration in Ros 2. In *IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 9692–9699. https://doi.org/10.1109/IROS47612.2022.9982082
[135] Mohit Mehndiratta and Erdal Kayacan. 2019. A Constrained Instantaneous Learning Approach for Aerial Package Delivery Robots: Onboard Implementation and Experimental Results. *Autonomous Robots* 43 (2019), 2209–2228.
[136] Xiaoli Meng, Heng Wang, and Bingbing Liu. 2017. A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LIDAR Sensor Fusion for Autonomous Vehicles. *Sensors* 17, 9 (2017), 2140. https://doi.org/10.3390/s17092140
[137] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: A Distributed Parallel Simulator for Multicores. In *Int'l Symp. on High-Performance Computer Architecture (HPCA)*. IEEE, 1–12. https://doi.org/10.1109/HPCA.2010.5416635
[138] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. 2016. Learning to Navigate in Complex Environments. *arXiv preprint arXiv:1611.03673* (2016). https://doi.org/10.48550/arXiv.1611.03673
[139] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. 2003. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping That Provably Converges. In *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, Vol. 3. 1151–1156. https://dl.acm.org/doi/10.5555/1630659.1630824
[140] Mircea Paul Muresan, Ion Giosan, and Sergiu Nedevschi. 2020. Stabilization and Validation of 3D Object Position Using Multimodal Sensor Fusion and Semantic Segmentation. *Sensors* 20, 4 (2020), 1110. https://doi.org/10.3390/s20041110
[141] Sean Murray, Will Floyd-Jones, George Konidaris, and Daniel J Sorin. 2019. A Programmable Architecture for Robot Motion Planning Acceleration. In *2019 IEEE 30th Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, Vol. 2160. IEEE, 185–188. https://doi.org/10.1109/ASAP.2019.000-4
[142] Sean Murray, William Floyd-Jones, Ying Qi, George Konidaris, and Daniel J Sorin. 2016. The Microarchitecture of a Real-Time Robot Motion Planning Accelerator. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 1–12. https://doi.org/10.1109/MICRO.2016.7783748
[143] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel J Sorin, and George Dimitri Konidaris. 2016. Robot Motion Planning on a Chip. In *Robotics: Science and Systems*, Vol. 6.
[144] Keiji Nagatani and Yozo Fujino. 2019. Research and Development on Robotic Technologies for Infrastructure Maintenance. *Journal of Robotics and Mechatronics* 31, 6 (2019), 744–751.
[145] Sabrina M. Neuman, Radhika Ghosal, Thomas Bourgeat, Brian Plancher, and Vijay Janapa Reddi. 2023. RoboShape: Using Topology Patterns to Scalably and Flexibly Deploy Accelerators Across Robots. In *Int'l Symp. in Computer Architecture (ISCA)* (FL) *(ISCA '23)*. ACM, NY, Article 69, 13 pages. https://doi.org/10.1145/3579371.3589104
[146] Sabrina M. Neuman, Brian Plancher, Thomas Bourgeat, Thierry Tambe, Srinivas Devadas, and Vijay Janapa Reddi. 2021. Robomorphic Computing: A Design Methodology for Domain-Specific Accelerators Parameterized by Robot

Morphology. In *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (ASPLOS '21)*. ACM, NY, 674–686. https://doi.org/10.1145/3445814.3446746

[147] Dima Nikiforov, Shengjun Chris Dong, Chengyi Lux Zhang, Seah Kim, Borivoje Nikolic, and Yakun Sophia Shao. 2023. RoSÉ: A Hardware-Software Co-Simulation Infrastructure Enabling Pre-Silicon Full-Stack Robotics SoC Evaluation. In *Int'l Symp. in Computer Architecture (ISCA)* (FL) *(ISCA '23)*. ACM, NY, Article 64, 15 pages. https://doi.org/10.1145/3579371.3589099

[148] Hiroki Ohta, Naoki Akai, Eijiro Takeuchi, Shinpei Kato, and Masato Edahiro. 2016. Pure Pursuit Revisited: Field Testing of Autonomous Vehicles in Urban Areas. In *Int'l Conf. on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. IEEE, 7–12. https://doi.org/10.1109/CPSNA.2016.10

[149] Jia Pan, Sachin Chitta, and Dinesh Manocha. 2012. FCL: A General Purpose Library for Collision and Proximity Queries. In *Int'l Conf. on Robotics and Automation (ICRA)*. IEEE, 3859–3866. https://doi.org/10.1109/ICRA.2012.6225337

[150] Jia Pan and Dinesh Manocha. 2012. GPU-Based Parallel Collision Detection for Fast Motion Planning. *The Int'l Journal of Robotics Research* 31, 2 (2012), 187–200. https://doi.org/10.1177/0278364911429335

[151] Luka Petrović, Juraj Peršić, Marija Seder, and Ivan Marković. 2020. Cross-Entropy Based Stochastic Optimization of Robot Trajectories Using Heteroscedastic Continuous-Time Gaussian Processes. *Robotics and Autonomous Systems* 133 (2020), 103618. https://doi.org/10.1016/j.robot.2020.103618

[152] Reid Pinkham, Shuqing Zeng, and Zhengya Zhang. 2020. Quicknn: Memory and Performance Optimization of K-D Tree Based Nearest Neighbor Search for 3d Point Clouds. In *2020 IEEE Int'l symposium on high performance computer architecture (HPCA)*. IEEE, 180–192. https://doi.org/10.1109/HPCA47549.2020.00024

[153] Jelena L Pisarov and Gyula Mester. 2021. The Use of Autonomous Vehicles in Transportation. *Tehnika* 76, 2 (2021), 171–177.

[154] Ira Pohl. 1970. Heuristic Search Viewed As Path Finding in a Graph. *Artificial intelligence* 1, 3-4 (1970), 193–204. https://doi.org/10.1016/0004-3702(70)90007-X

[155] Pengju Ren, Mieszko Lis, Myong Hyon Cho, Keun Sup Shim, Christopher W Fletcher, Omer Khan, Nanning Zheng, and Srinivas Devadas. 2012. Hornet: A Cycle-Level Multicore Simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 31, 6 (2012), 890–903. https://doi.org/10.1109/TCAD.2012.2184760

[156] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. 2021. Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding. In *Int'l Conf. on Computer Vision (ICCV)*. https://doi.org/10.48550/arXiv.2011.02523

[157] Nezam Rohbani, Sina Darabi, and Hamid Sarbazi-Azad. 2021. PF-DRAM: A Precharge-Free DRAM Structure. In *Int'l Symp. in Computer Architecture (ISCA)*. IEEE, 126–138.

[158] Jacob Sacks, Divya Mahajan, Richard C Lawson, and Hadi Esmaeilzadeh. 2018. RoboX: An End-To-End Solution to Accelerate Autonomous Control in Robotics. In *Int'l Symp. in Computer Architecture (ISCA)*. IEEE, 479–490. https://doi.org/10.1109/ISCA.2018.00047

[159] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. 2018. PythonRobotics: A Python Code Collection of Robotics Algorithms. *arXiv preprint arXiv:1808.10703* (2018). https://doi.org/10.48550/arXiv.1808.10703

[160] Yuki Sakata and Takuo Suzuki. 2023. Coverage Motion Planning Based on 3D Model's Curved Shape for Home Cleaning Robot. *Journal of Robotics and Mechatronics* 35, 1 (2023), 30–42.

[161] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems. In *Int'l Symp. in Computer Architecture (ISCA)*. https://doi.org/10.1145/2508148.2485963

[162] Stefan Schaal. 2006. Dynamic Movement Primitives-A Framework for Motor Control in Humans and Humanoid Robotics. In *Adaptive Motion of Animals and Machines*. Springer, 261–280. https://doi.org/10.1007/4-431-31381-8_23

[163] Tilmann Schlenther, Kai Martins-Turner, Joschka Felix Bischoff, and Kai Nagel. 2020. Potential of Private Autonomous Vehicles for Parcel Delivery. *Transportation Research Record* 2674, 11 (2020), 520–531.

[164] Cornelia Schulz and Andreas Zell. 2020. Real-Time Graph-Based SLAM with Occupancy Normal Distributions Transforms. In *Int'l Conf. on Robotics and Automation (ICRA)*. IEEE, 3106–3111. https://doi.org/10.1109/ICRA40945.2020.9197325

[165] Brian C Schwedock, Piratach Yoovidhya, Jennifer Seibert, and Nathan Beckmann. 2022. Täkō: A Polymorphic Cache Hierarchy for General-Purpose Optimization of Data Movement. In *Int'l Symp. in Computer Architecture (ISCA)*. 42–58. https://doi.org/10.1145/3470496.3527379

[166] Deval Shah, Ningfeng Yang, and Tor M. Aamodt. 2023. Energy-Efficient Realtime Motion Planning. In *Int'l Symp. in Computer Architecture (ISCA)* (FL) *(ISCA '23)*. ACM, NY, Article 57, 17 pages. https://doi.org/10.1145/3579371.3589092

[167] Shihao Shen, Yilin Cai, Jiayi Qiu, and Guangzhao Li. 2022. Dynamic Dense RGB-D SLAM Using Learning-Based Visual Odometry. *arXiv preprint arXiv:2205.05916* (2022).

[168] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. 2004. Manipulation Planning with Probabilistic Roadmaps. *The Int'l Journal of Robotics Research* 23, 7-8 (2004), 729–746. https://doi.org/10.1177/0278364904045471

[169] Vadym Slyusar, Mykhailo Protsenko, Anton Chernukha, Vasyl Melkin, Oleh Biloborodov, Mykola Samoilenko, Olena Kravchenko, Halyna Kalynychenko, Anton Rohovyi, and Mykhaylo Soloshchuk. 2022. Improving The Model Of Object Detection On Aerial Photographs And Video In Unmanned Aerial Systems. *Eastern-European Journal of Enterprise Technologies* 1, 9 (2022), 115.

[170] Stephen Somogyi, Thomas F. Wenisch, Anastasia Ailamaki, and Babak Falsafi. 2009. Spatio-Temporal Memory Streaming. In *Proceedings of the 36th Annual Int'l Symp. on Computer Architecture* (TX) *(ISCA '09)*. ACM, NY, 69–80. https://doi.org/10.1145/1555754.1555766

[171] Statista Research Department. 2021. Global Robotics Market Revenue 2018–2025. https://www.statista.com/statistics/760190/worldwide-robotics-market-revenue/.

[172] Sebastian Thrun. 2002. Probabilistic Robotics. *Commun. ACM* 45, 3 (2002), 52–57.

[173] Inam Ullah, Xin Su, Xuewu Zhang, and Dongmin Choi. 2020. Simultaneous Localization and Mapping Based on Kalman Filter and Extended Kalman Filter. *Wireless Communications and Mobile Computing* 2020 (2020), 2138643:1–2138643:12. https://doi.org/10.1109/SIU.2009.5136492

[174] Jingyao Wang, Manas Ranjan Pradhan, and Nallappan Gunasekaran. 2022. Machine Learning-Based Human-Robot Interaction in ITS. *Information Processing & Management* 59, 1 (2022), 102750.

[175] Xuewu Wang, Xin Zhou, Zelong Xia, and Xingsheng Gu. 2021. A Survey of Welding Robot Intelligent Path Optimization. *Journal of Manufacturing Processes* 63 (2021), 14–23.

[176] Yifan Wang, Long Zhang, and Gang Chen. 2019. Optimal Sensor Placement for Obstacle Detection of Manipulator Based on Relative Entropy. In *2019 14th IEEE Conf. on Industrial Electronics and Applications (ICIEA)*. IEEE, 702–707. https://doi.org/10.1109/ICIEA.2019.8833986

[177] John T Wen and Steve H Murphy. 1990. PID Control for Robot Manipulators. (1990).

[178] Thomas Whelan, Stefan Leutenegger, Renato Salas-Moreno, Ben Glocker, and Andrew Davison. 2015. ElasticFusion: Dense SLAM without a Pose Graph. Robotics: Science and Systems. https://doi.org/10.1177/0278364916669237

[179] Ruolin Ye, Wenqiang Xu, Haoyuan Fu, Rajat Kumar Jenamani, Vy Nguyen, Cewu Lu, Katherine Dimitropoulou, and Tapomayukh Bhattacharjee. 2022. RCare World: A Human-Centric Simulation World for Caregiving Robots. In *2022 IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 33–40.

[180] Ayesha Younis, Li Shixin, Shelembi Jn, and Zhang Hai. 2020. Real-Time Object Detection Using Pre-Trained Deep Learning Models MobileNet-SSD. In *Int'l Conf. on Computing and Data Engineering (ICCDE)*. 44–48. https://doi.org/10.1145/3379247.3379264

[181] Bo Yu, Wei Hu, Leimeng Xu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. 2020. Building the Computing System for Autonomous Micromobility Vehicles: Design Constraints and Architectural Optimizations. In *Int'l Symp. on Microarchitecture (MICRO)*. IEEE, 1067–1081. https://doi.org/10.1109/MICRO50266.2020.00089

[182] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, and Srinivas Devadas. 2015. IMP: Indirect Memory Prefetcher. In *Int'l Symp. on Microarchitecture (MICRO)*. 178–190. https://doi.org/10.1145/2830772.2830807

[183] Qi-bin Zhang, Peng Wang, and Zong-hai Chen. 2019. An Improved Particle Filter for Mobile Robot Localization Based on Particle Swarm Optimization. *Expert Systems with Applications* 135 (2019), 181–193. https://doi.org/10.1016/j.eswa.2019.06.006

[184] Hengyu Zhao, Yubo Zhang, Pingfan Meng, Hui Shi, Li Erran Li, Tiancheng Lou, and Jishen Zhao. 2019. Towards Safety-Aware Computing System Design in Autonomous Vehicles. *arXiv preprint arXiv:1905.08453* (2019). https://doi.org/10.48550/arXiv.1905.08453

[185] Jianfeng Zheng, Shuren Mao, Zhenyu Wu, Pengcheng Kong, and Hao Qiang. 2022. Improved Path Planning for Indoor Patrol Robot Based on Deep Reinforcement Learning. *Symmetry* 14, 1 (2022), 132.

[186] Yichao Zhou and Jianyang Zeng. 2015. Massively Parallel A* Search on a GPU. In *Proceedings of the AAAI Conf. on Artificial Intelligence* (Austin, Texas) *(AAAI'15)*. AAAI Press, 1248–1254. https://doi.org/10.1609/aaai.v29i1.9367

Received August 2023; revised October 2023; accepted October 2023