Towards computational awareness in autonomous robots: an empirical study of computational kernels

Ashrarul H. Sifat1 · Burhanuddin Bharmal1 · Haibo Zeng1 · Jia-Bin Huang2 · Changhee Jung3 · Ryan K. Williams1

Received: 24 June 2022 / Accepted: 20 March 2023 / Published online: 3 May 2023
© The Author(s) 2023

Abstract
The potential impact of autonomous robots on everyday life is evident in emerging applications such as precision agriculture, search and rescue, and infrastructure inspection. However, such applications necessitate operation in unknown and unstructured environments with a broad and sophisticated set of objectives, all under strict computation and power limitations. We therefore argue that the computational kernels enabling robotic autonomy must be scheduled and optimized to guarantee timely and correct behavior, while allowing for reconfiguration of scheduling parameters at runtime. In this paper, we consider a necessary first step towards this goal of computational awareness in autonomous robots: an empirical study of a base set of computational kernels from the resource management perspective. Specifically, we conduct a data-driven study of the timing, power, and memory performance of kernels for localization and mapping, path planning, task allocation, depth estimation, and optical flow, across three embedded computing platforms. We profile and analyze these kernels to provide insight into scheduling and dynamic resource management for computation-aware autonomous robots. Notably, our results show that there is a correlation of kernel performance with a robot’s operational environment, justifying the notion of computation-aware robots and why our work is a crucial step towards this goal.

Keywords Computational intelligence · autonomous robots · intelligent scheduling · complex systems

Introduction
Robots are demonstrating the ability to tackle real-world problems if the problems are well-defined and composed of a reasonably small, fixed set of high-level tasks, e.g., welding on an assembly line [1]. However, our understanding remains limited for autonomous robots that solve complex problems composed of a wide range of difficult computational tasks. Consider for instance a search and rescue problem where autonomous robots must actively collaborate with human searchers in complex, uncertain environments (e.g., wilderness or disaster sites). In such a problem, an autonomous robot will be expected to execute (at least) the following base set of computational kernels: simultaneous mapping and localization [2]; path planning and navigation [3]; high-level decision-making such as task allocation [4]; and a range of computer vision (CV) tasks such as optical flow [5] and depth estimation [6]. Furthermore, the remoteness of the operational environment necessitates on-board computation by the robots, placing strict limits on computation and power. Finally, as the robots work in concert with humans, there must exist guarantees on robot behavior, and thus computation, to ensure safety. We argue that the search and rescue example motivates the need for robots that are computation-aware, allowing the computational kernels that support autonomy to be scheduled and optimized on embedded hardware in a manner that adapts to changing objectives and uncertain environments, while guaranteeing the timeliness and correctness of autonomous behavior.

Towards this vision of computational awareness in autonomous robots, we must first understand the computational requirements for the base set of kernels that support robotic autonomy (listed above). In the current literature, there is a deep understanding of the aforementioned computational kernels from a theoretical point of view, for example
yielding insights into worst-case computational complexity [7–9]. However, given a particular computational platform (such as the NVIDIA Xavier [10] studied in this work), there is very little existing work on the relationship between the high-level tasks an autonomous robot performs and the underlying computational requirements for performing such tasks in a timely manner. Additionally, there have been recent efforts to integrate both hard and soft real-time scheduling-capsable kernels with the Robotic Operating System (ROS) [11, 12], which can form the basis for guaranteeing the timeliness and correctness of autonomous behaviors, and with ROS 2.0, real-time scheduling support specialized for robotic applications is in sight [13]. It is therefore critical that we build an understanding of computational kernels for autonomous robots so that such capabilities can be fully exploited in the near future. Thus, this paper performs an empirical study of the computational requirements of a base set of kernels depicted in Fig. 1, which is the first step achieving computational awareness in autonomous robots.

The purpose and contribution of the paper are summarized in the following lines:

1. A survey of the most popular as well as SOTA robotic computational kernels for a base set of applications, including SLAM, path planning, task allocation, and depth and optical flow estimation “Survey of Computational Kernels”.
2. Groundwork for computation aware autonomy via comprehensive data analysis of the computational kernels for execution times, utilization, power, and memory consumption “Empirical Analysis”.
3. Deduction of dynamic environmental correlation of computational kernels for SLAM and path planning which supports the need for the data and analysis provided in this paper “Discussions”.
4. Insight drawn from the data, to understand the impact of scheduling and optimization pointing to solutions for guaranteeing system timeliness “Discussions”.
5. Release of geotagged indoor/outdoor dataset with LIDAR pointcloud, GPS and IMU data as well as aforementioned computational benchmarking data for utilization in robot deployment with dynamic scheduling https://github.com/caslab-vt/CAA-profiler.
6. Additional insights on CPU-GPU interaction of Deep Learning based CV kernels in heterogeneous hardware and discussion on scope of future research “Discussions”.

Related work While there is very little existing work on computational awareness in autonomous robots, we can point to a few recent efforts that are related to this paper. A recent work focuses on benchmarking connected and autonomous vehicles (CAVs) applications from the quality of service (QoS) perspective [14]. The benchmarking done in this work is solely concentrated on CAVs and derive insights such as the need for heterogeneous architecture, high memory bandwidth and optimized cache architecture, which we also recognise in our systems. However, it lacks the in-depth analysis of each computational kernel that might need to run in parallel in a mobile robot unlike the CAVs and can have a different wide range of parameters to consider. Similarly, a specialized benchmark for mobile vision algorithms is presented in [15], which provides a useful micro-architectural analysis of vision kernels. However, it also lacks the insight into the bigger spectrum of robotic computational kernels as well as environmental correlation and kernel specific parametric study. Kato et al. [16] present Autoware on board, to integrate the Autoware open source software for autonomous
driving [17] on the Nvidia Drive Px2 and subsequently measure the execution time performance. However, this work lacks sufficient understanding on how the computation correlates with the driving environment. Zhao et al. [18] perform a field study for autonomous vehicles (AV) to gain insights on how the computing systems should be designed. This work looks into the sequential operation of the AV computational kernels from a safety perspective and present a innovative safety criteria for AV. Both of these works [16, 18] are solely concentrated on AV which have different constraints than mobile robots such as an Unmanned Aerial Vehicle (UAV). Finally, Carlone and Karaman [19] consider the visual-inertial navigation problem, and propose anticipation with environmental cues to build a simplified model on robot dynamics and produce a computationally efficient system. However, it is an analysis of just one kernel, naturally not enough to implement a general framework such as our proposed solution. All in all these literature give valuable insight into the need for computational awareness and specific studies but not the comprehensive analysis for autonomous robotic platforms.

Our focus is on robotic systems research that have shed light upon computational awareness. Recently, ROS 2.0 has been deployed to solve the problems of computation and network congestion impacts on the communication latencies [20]. We are also inspired by recent insights into the cognitive map, a hypothesis that the human brain builds a unified representation of the spatial environment to support memory and guide future action [21]. There is also work on architecture-aware algorithmic approach to reduce data movement and the resulting latency and power consumption [22].

The concept of computational awareness is popular in the real-time community [23–26]. The problem of dynamic scheduling in heterogeneous multiprocessor hardware can be addressed from many perspectives as can be seen in these literature. However there is very little existing work in robotics which focus on dynamic scheduling for robotics systems. This is the clear gap between the real-time community and robotics, as robotics research is most often concerned with developing, improving, and deploying individual kernels. Instead, this paper recognizes that robots applied to real problems must run various kernels in parallel and with safety as a key requirement. In such cases, the data profiles given in this paper, and appropriate real-time scheduling based on this data, are a clear requirement.

Finally, the example of cloud robotics, in some sense, can be interpreted as the robotic systems allowing computational awareness to offload computation [27–29]. However, it is a much different system than what we aim for, as we recognize the scenario when offloading to the cloud is simply not possible for many reasons.

**Preliminaries**

Autonomous robots continue to suffer from strict limits on computation and power, especially for small aerial vehicles. From a resource management perspective, we therefore focus on profiling that can inform two primary techniques: (1) thread-level scheduling that prioritizes among multiple computational kernels competing for shared computational resources such as CPU, GPU, and memory [30–33]; and (2) the Dynamic Voltage and Frequency Scaling (DVFS) mechanism on modern processors that allows for fine-grained frequency scaling and power management [34]. We aim to analyze each computational kernel for sustainability and predictability by showing their worst case behavior, average behavior, and the variation from them. For an autonomous robot, we argue that a static, single-valued worst case execution time (WCET) is too pessimistic as the robot’s computational needs are highly dynamic and potentially dependent on the environment and goal. To take advantage of the above techniques in a robotics setting, we profile the utilization, timing, memory, and power consumption properties of a base set of computational kernels that support autonomy.

There are several different parameters to consider when selecting the most appropriate scheduling algorithm. Figure 2 depicts the general operation of a scheduler, with definitions for the parameters we aim to measure given below.

- **CPU Utilization**: The ratio of a CPU’s load and the computational capacity of the CPU.
- **Throughput**: The total number of processes completed per unit time, i.e., the total amount of work done in a unit of time.
- **Turnaround/Response Time**: The amount of time taken to execute a particular process/thread.
- **User Time**: The amount of time a process/thread spends in the user space of the host operating system. This is also referred to as CPU execution time.
- **System Time**: The amount of time a process spends in kernel level instructions such as interrupts, I/O calls, etc.
- **Activation Period**: The interval or period after which each periodic kernel is repeated and therefore dispatched to the CPU scheduler.
In general, CPU utilization and throughput are maximized and other factors are minimized to achieve performance optimization. While the response time is varied widely across different systems and scheduling policies, the execution time is tied most closely to the specifics of hardware and the computational kernel itself. Therefore, in this paper we focus our profiling on the execution time in user space and kernel space as a means of characterizing the requirements for our computational kernels. At the same time, thread-level execution time breakdown and CPU-affinity are also a necessary part of scheduling. Therefore, among the other parameters of analysis we will provide the number of processes per computational kernel, number of threads per process, average utilization of threads, memory utilization, and power consumption.

We also investigate relevant parameters for GPU-intensive vision kernels, including instructions per cycle (IPC), shared utilization, various (xyz) functional unit utilization, achieved occupancy, and various memory load throughput. As the first step towards optimizing and guaranteeing the timeliness of computation for an autonomous robot, we believe these parameters are of vital importance to understand. GPU performance parameters also vary significantly across different architectures. The definitions of the parameters we measure are given below:

- **Achieved Occupancy**: Ratio of the average active warps per active cycle to the maximum number of warps supported on a GPU.
- **xyz FU Utilization**: The utilization level of the GPU xyz functional units on a scale of 0 to 10. Here xyz depends on the hardware platforms. Typically xyz = alu, cf, special etc.
- **Warp Execution Efficiency**: Ratio of the average active threads per warp to the maximum number of threads per warp supported on a GPU.
- **Shared Utilization**: The utilization level of shared memory relative to peak utilization on a scale of 0 to 10.
- **GPU Utilization**: Percent of time over the past sample period during which one or more kernels was executing on the GPU. (Only for Zotac platform using Nvidia-smi interface).
- **GPU Memory Usage**: Total memory allocated by active contexts. (Only for Zotac platform using Nvidia-smi interface). The GPU memory utilization can be derived by scaling this value (%) to the GPU memory capacity.

Finally, it is worth noting that the computational complexity of the tasks carried out by an autonomous robot can be used to *mathematically* formulate the execution time required for each CPU process. However, the complex nature of CPU execution cycles combined with the hardware and kernel related variables and dependencies makes it infeasible to predict computational needs merely from a mathematical formulation. Hence, we begin in this paper with profiling of our computational kernels in various settings across popular computational platforms for robotics. Only then is it possible to answer the pressing questions of combining computational kernels, implementing appropriate scheduling policies, and determining the rates at which we can execute the kernels.

**Survey of computational kernels**

This section provides a brief overview of the concept and operations of the computational kernels we profile. Note that *each computational kernel may be broken into multiple threads or even processes*, as detailed in “Empirical Analysis”. We provide a concise but relevant survey of our base set of computational kernels that represent robotic autonomy in a broader scale. It is to be noted here that this is *not* the total spectrum of robotic computational algorithms and tasks. Our objective is to focus on a base set of kernels for laying the groundwork for computational awareness in robotic autonomy. Any robot by definition of autonomy has to navigate in the environment using some version of SLAM and path planning, it has to perceive the environment using kernels such as object detection, optical flow and depth estimation, and it has to allocate a broad set of high-level tasks according to mission objectives. We try to include all of these criteria into our survey by citing the most relevant computational kernels that support these autonomous behaviors. Our key claim is that any autonomous robot moving forward must run such kernels at a minimum to achieve any level of autonomy, and thus there is significant survey value in our work.

**Simultaneous localization and mapping**

Simultaneous Localization and Mapping (SLAM) is an age-old problem in robotics which still attracts significant interest from the robotics community. SLAM enables a robot to build an understanding of its environment over time and is thus foundational to robotics applications (see the left hand side of Fig. 1 for a depiction of typical SLAM output). In this work, we select three point cloud-based methods for SLAM: Gmapping [35], Lidar Odometry and Mapping (LOAM) [36] and Google Cartographer [37] (which are commonly applied in robotics), as well as one vision based SLAM kernel called ORBSLAM3 [38]. We use four versions of SLAM to demonstrate two related concepts: (1) computational kernels for a given task can vary significantly in performance; and (2) different kernel versions may be used interchangeably based on computational/power limitations and robot objectives.

As one of the most popular computational kernels for outdoor SLAM, Gmapping consists of the Rao-Blackwellized particle filter to learn grid maps from laser range data. This
approach uses a particle filter in which each particle carries an individual map of the environment. It is one of the pioneering approaches to SLAM [2], with a tractable computational complexity of $O(NM)$, where $N$ is the number of particles and $M$ is the size of the corresponding grid map. We also profile LOAM which produces real-time 3D maps from Lidar point cloud data, which are generally of a higher quality than Gmapping. LOAM is a real-time method for odometry and mapping using range measurements from a 2-axis Lidar moving in 6-DOF. The problem is hard because the range measurements are received at different times, and errors in motion estimation can cause mis-registration of the resulting point cloud. Typically, coherent 3D maps can be built by off-line batch methods, often using loop closure to correct for drift over time. This method achieves both low-drift and low-computational complexity without the need of high accuracy ranging or inertial measurements. LOAM has four parallel process nodes running simultaneously to update the mapping and odometry data and produces updates on maps in real-time (see Fig. 3).

**Remark 1** It is important to note that the generation of maps and localization solutions is highly dependent on the features of the environment. We will demonstrate that this phenomenon can be seen directly in the computational profiling for SLAM, where we contrast between indoor and outdoor environments in terms of execution times and CPU usage.

We also investigate a third pointcloud based and SOTA SLAM kernel, the Google Cartographer to demonstrate a full range of LIDAR based SLAM kernel [37] and their comparative profiles. We also implement it in ROS environment which supports efficient transfer of transformation(/tf) data. The cartographer has two main subsystems or processes i.e. Local SLAM and Global SLAM. The local SLAM is the heaviest process by far which produces a succession of local submaps. The Global Slam runs at a slower frequency and tie the submaps most consistently together, looking for loop closure detection. A third lightweight process is publishing the robot states as tf transforms which is common in other SLAM kernels. Figure 4 depicts the Cartographer SLAM in ROS environment [39].

Finally we investigate a visual SLAM of very recent development known as ORBSLAM3 [38]. The main feature of ORBSLAM3 is a tightly integrated visual-inertial SLAM system that fully relies on maximum a posteriori (MAP) estimation, even during IMU initialization. This results in real-time robust operation in small and large, indoor and outdoor environments. This SLAM however is less intensive on the memory as it produces sparse maps compared to the other methods. It is the state-of-the-art in visual SLAM and a base kernel for more complex SLAM systems such as Dynamic ORBSLAM [40] and DYNASLAM [41].

Figure 5 refers to the internal system of ORBSLAM3 which comprises of several building blocks. The main blocks are (1) Atlas, a multimap representation composed of a set of disconnected maps tracking with DBoW2 keyframe database, (2) a tracking thread that processes sensor information and computes the pose of the current frame with respect to the active map in real time, (3) local mapping that adds keyframes and points to the active map, removes the redundant ones, and refines the map using visual or visual-inertial Bundle Adjustment, and (4) finally a loop and map merging block that detects common regions between the active map and the whole Atlas at keyframe rate. ORBSLAM3 is designed to work with a vast number of camera types, however in our experiments, we investigate the RGB-D TUM dataset [42].

**Path planning**

Path planning is another critical computational kernel for autonomous robots (see Fig. 1 left for a depiction of typical
planning methods based on Rapidly-Exploring Random Trees (RRTs) [43], as they are very popular and there are numerous RRT variants for various application scenarios. We profile a 2D RRT method which utilizes R-tree [44] to improve performance by avoiding point-wise collision and distance checking. When profiling the RRT kernel, we generate an environment with random obstacles to represent dynamic and unknown environments that the robot is presumed to traverse. Importantly, the execution time of RRT will depend on the size of the environment, the number of planning iterations allowed, the goal location to reach, obstacles, and the step or sampling size of the planner. In this work, we investigate the effect of step/sampling size (R), environment size, and obstacle density on RRT computational requirements.

In addition, we investigate the effect of parallelizing the RRT kernel. There is a notable work on utilizing GPU multithreading capabilities such as CUDA cores that are available in Nvidia Jetson hardware platforms [45]. This work focuses on parallelizing the collision-checking procedure, which is generally recognized as the computationally expensive component of sampling-based motion planning algorithms. Importantly, RRT scales to arbitrarily high dimensionality. RRT therefore allows path planning to occur in multidimensional spaces with obstacle avoidance. This can be applied to physical obstacles in space or operational obstacles such as singularities in the kinematics of a robot. RRT is a perfect kernel for parallelization because it is physical system agnostic and all collision checking happens in parallel. Effectively parallelizing this algorithm would allow robots to more rapidly explore spaces, which can have many positive downstream effects. The naïve implementation of a massively parallelized RRT with CUDA did yield performance benefits when the obstacle count was greater than 2048 as observed by the authors. However, the RRT algorithm and CUDA do not mesh perfectly due to the nature of the parallelizable aspect of the RRT, the collision checking, also having a dependence on spatial locality [45].

**Task allocation**

Single robots and multi-robot teams often have multiple high-level tasks to complete over a mission, with the tasks potentially streaming in or changing in nature over time (see the left hand side of Fig. 1 for a depiction of task allocation). To model the autonomous decision-making necessary to allocate robot resources to tasks optimally, we profile task allocation problems which are difficult combinatorial optimization problems with resource constraints. In this paper, we benchmark and analyze a novel task allocation problem that aims to allocate a set of robots’ functionalities to a mission such that certain functionality combinations are satisfied. A combinatorial optimization problem with matroid constraints is solved using a greedy algorithm, yielding a solution with bounded suboptimality [4, 46]. The parameters of interest for this task allocation kernel are the robot cardinality, $N$, requirement cardinality $R$, and functionality cardinality $F$. By changing these parameters, we can alter the complexity of the task allocation problem and profile its computational requirements.

**Computer vision (CV) kernels**

To support a robot navigating an environment safely while avoiding *dynamically moving objects* (such as pedestrians), we study two additional kernels for GPU-intensive computer vision tasks (see Fig. 1 left for a depiction of the typical input/output). Specifically, in this work, we focus on 1) optical flow (for estimating dense motion between consecutive frames) and 2) monocular depth estimation (for predicting dense scene depth). In each kernel, we choose the state-of-the-art models of PWC-Net [5] for optical flow and MiDaSv2 [6] as well as a depth estimation based on fastdepth [47].

In MiDaS, the authors introduce the dense prediction transformer. It is a dense prediction architecture based on an encoder-decoder design that leverages a transformer as the basic computational building block of the encoder [6]. Unlike fully-convolutional networks, their vision transformer backbone foregoes explicit downsampling operations after an initial image embedding has been computed and maintains a representation with constant dimensionality throughout all processing stages. Moreover, it has a global receptive field at every stage. It is shown by the authors that these properties are especially advantageous for dense prediction tasks as they naturally lead to fine-grained and globally coherent predictions. Monocular depth estimation is typically cast as a dense regression problem. This kernel can predict both monocular depth estimation and Semantic Segmentation.

For optical flow estimation, the compact but effective PWC-Net has been designed to make full use of the simple and well-established principles. It makes significant improvements in model size and accuracy over existing CNN models for optical flow. It is shown to outperform all published optical flow methods on the MPI Sintel final pass and KITTI 2015 benchmarks [5]. In our experiments for both models, we use the publicly available pre-trained models in PyTorch for our empirical analysis.

The authors of fastdepth [48] have designed a fast model for monocular depth estimation. As compared to the most used and accurate model, Midasv2, developed by Intel ISL [6], the fastdepth model is fast and is designed especially for an embedded device with constrained resources. For our experimentation purpose, we have used the pre-trained model provided by the [48] and built our model over it. We have extended the same model in the darknet framework, with a minor change in activation function from ReLU to Leaky
ReLU. We call this model fastdepthv2. The Leaky ReLU and its variant work better than the ReLU activation function [49]. A notable difference with the previous two CV kernels is that this one is a C library as opposed to the python library for the previous two.

Empirical analysis

In this section we present the empirical analysis on the chosen set of computational kernels and the supporting architectural platforms on the criteria we discuss in “Preliminaries”. We describe the profiling mechanism, and provide the environmental correlation of SLAM and path planning as well as the primary insights for different platforms utilization of the kernels. This is the groundwork for computation aware autonomy as these data of timing, utilization, power and memory consumption is pivotal to scheduling robotic computational kernels in embedded heterogeneous platforms which we discuss in “Discussions”. Our objective in this paper is to present this comprehensive data analysis leading to the possible issues a robotic computational pipeline can face with some insights on how those issues can be potentially solved.

Mobile robot platform

As our motivation in this work is to deploy computation-aware robots to solve real-world problems, we aimed to exploit real-world data for profiling where possible. Sensor data collection for several computational kernels was conducted using the Jackal Unmanned Ground Vehicle (UGV) platform from Clearpath robotics [50]. The UGV was equipped with the Velodyne VLP-16 LIDAR and a GoPro Fusion 360 camera, with the sensors mounted as seen in Fig. 6.

![Fig. 6 Platform for data collection. Velodyne LIDAR and Go-Pro Fusion 360 camera are mounted on a Jackal UGV](image)
Computational platforms

We consider three representative hardware platforms for robots. Their technical specifications are summarized in Table 1. Our first platform is the NVIDIA Jetson AGX Xavier [10] which has significant computational capacity for autonomous robots in a compact form factor. Additionally, the Xavier has a heterogeneous architecture based on a System-on-Chip (SoC) that supports both CPU and GPU computation. The second computational platform we consider is the NVIDIA Jetson Xavier NX [51] which is the state-of-the-art embedded platform from NVIDIA that is suited for drones and smaller robots with strict resource constraints. Both Xavier devices have shared memory between the CPU and GPU which along with their architecture makes them extremely power efficient. Finally, we profile our kernels on the Zotac Mini PC system with specifications similar to a desktop computer, but with a form factor that allows integration into a larger robot with high power availability.

Remark 2 We have tried to conform to the most widely used as well capable hardware for robotics today. And we are seeing that many industrial robots such as delivery robots are utilizing the NVIDIA Jetson platforms. Therefore our main testbench comprises of the Nvidia hardware. As seen in Table 1, the CPU and GPU capabilities of these hardware platforms, even at such a low power consumption, are unparalleled compared to other robotics hardware platforms. Nevertheless, it is valuable to make a comparison to more general hardware such as the Intel Zotac platform. Hence, we also consider the Intel NUC and the Gigabyte Brix, which are used in many wheeled as well as humanoid robots. However, these platforms do not have the combination of CPU and GPU computation that we want to achieve. With the influx of GPU intensive computation of recent times, we wanted to give a detailed view of that spectrum. Therefore we chose the Intel Zotac platform which albeit a highly power consuming platform has the capabilities of the Nvidia hardware. It has the discrete GPU which is the reason of higher form factor and power consumption. Other than the GPU, it is quite similar to Intel NUC and Gigabyte Brix.

Moreover, although the Nvidia AGX and NX share a similar architecture, we have shown in Table 1 that they are still very different in computing capabilities. The Jetson Xavier AGX form factor is more suitable to ground robots whereas the NX is a perfect fit for aerial vehicles. The power consumption is also significantly higher in AGX. As a result, AGX is a more capable platform. It is worth finding out which kernels are reasonable to run in AGX vs. NX, a critical mission information to know before designing the entire software system.

From the software perspective, the Linux environment is the most popular open source operating system used for robotics worldwide. Linux coupled with ROS is the go-to system for robotics researchers. Thus, we profile our platforms/kernels in Ubuntu 18.04 with ROS middleware (ROS Melodic). The kernel version of Linux for the Xavier platforms is 4.9.140-tegra which is the latest ARM-compatible version of the vanilla Linux kernel. For the Zotac platform, we use the Linux kernel 5.4.

Profiling data acquisition

For CPU execution time, utilization, and shared memory usage, we primarily use psutil [52]. Psutil is a python tool that implements many basic Linux command-line operations such as ps and uptime, among others. However, for the task allocation kernel, CPU execution time is calculated using the times Linux kernel api, CPU utilization is determined using mpstat, and the vmstat command provides memory usage. The process and threads details for kernels were captured using NVIDIA Nsight Systems whereas GPU statistics were recorded using the NVIDIA Profiler tool [53] for NVIDIA Xaviers. For the Zotac platform, GPU memory usage, GPU utilization, and GPU power were recorded using the gpustat command which is based on the Nvidia-smi interface.

For power measurements, on-board INA power monitors in the Nvidia platforms are utilized which allow power monitoring at runtime with an accuracy of 95%. For the Zotac platform, we calculate CPU power usage with the energyusage python command which calculates the power via the RAPL (Running Average Power Limit) interfaces found on Intel processors [54].

The objective of this paper is to gain insight into the computational aspects of crucial robotic kernels. Therefore, we use these well known and rigorous tools developed over a long time and reputation, so that our results are not questionable. Developing custom tools is not in the scope of this paper, but can certainly be pursued when the optimization problem is narrowly defined such as in RESCH [55].

Profiling results

In this section, we present our profiling results with only limited initial insights. The implications and usefulness of these results are discussed in detail in “Discussions”. As a reference, we provide a recorded video of the data collection experimentation as a recorded video https://youtu.be/QQ8IeRK0PgE.

SLAM

The LOAM SLAM benchmark was carried out with the real-world data collected as described previously. For the data presented in this paper, the same loop in a large, mostly outdoor area was traversed three times as is seen in Fig. 7. During this traversal, the first completion of the loop did not achieve
loop closure in the SLAM process (feature reconciliation that improves maps as more data are collected), whereas the second and third loop completion did see loop closure. As a result, it is visible that the execution time decreases rapidly between the second and third loop which is visible in the computational profile (Fig. 8). Moreover, there was an average of 30% drop in execution time when passing through indoor confined spaces such as a tunnel. Finally, in Fig. 8a we see that the RAM utilization increases with the traversal of the environment. Based on the above discussion, we can deduce that repeated environment traversal, environment complexity (indoor vs. outdoor), and mission length correlate strongly with the computational requirements for SLAM.

Remark 3 Regarding the explanation of the memory consumption, it is highly unlikely that it has a memory leak, at least from primary inspection of the code as memory allocation was not used in it. LOAM SLAM [36] is a very well cited and high quality LIDAR based 3D SLAM technique which has high load on the processor. It has 16 threads (4 x 4) and 4 dominant threads from the four processes. Moreover, the scope of this paper is not to look into particular coding issues but to identify computational and environmental aspects in robotics kernels. As we do not see this attribute in the 2D SLAM discussed later, the memory buildup is likely a characteristic of 3D high definition LOAM SLAM. However, further micro-architectural analysis into this kernel would be necessary to pin down this attributes.

Table 2 summarizes the different processes, thread counts, and percentages of execution times for the LOAM kernel in the Xavier platform. To specify more clearly, this data in this table are generated using the Nvidia profiler and its definitions are a little different than the ones presented in section II. Here, The % utilization of dominant thread denotes % utilization of a single core, the % of Execution time denotes the percentage of the particular process’s execution time compared to the total execution times of these four process that make up the LOAM kernel and the average execution time is just the average of that process. SLAM is highly dependent on sensor data and thus the ROS middleware is used to enable smooth message passing between the different SLAM processes. The Laser Mapping and Odometry processes contribute towards the map building for LOAM and thus require the most resources combined. On the other hand, the Scan Registration process only contributes towards point cloud data processing which is a constant cycle and therefore has very little variation in data. Another vital aspect is that all of these processes have one dominant thread for scheduling purposes as the other threads can be regarded as overhead and therefore makes it convenient from a scheduling perspective. From the tracing profiles in Nsight Systems, it is visible that these four processes run in parallel in different CPU cores while the ROS middleware ensures the message passing between them. Because of ROS’s lack of quality of service (QoS) in communication and real-time support, some momentary drops in packets are observed when the data of CPU performance were collected during the kernel’s operation.

Figures 9 and 10 depict the LOAM performance metrics in the Xavier NX and Zotac platforms, respectively. It is visible that the execution times are mostly the same for Xavier NX but the CPU and memory utilization are significantly higher which is expected behavior. However, in the Zotac platform the execution times are significantly lower than the Xavier platforms which is also expected considering its higher frequency and higher CPU and memory capacity.

Moving to the Gmapping SLAM kernel, which also uses our real-world data, it is evident from the profile in Fig. 11 that the variation is far less when compared to LOAM and it has a lower utilization and execution time for the Xavier AGX platform. However, due to the powerful computational capabilities of Zotac, we don’t see this difference between LOAM and Gmapping in Zotac. The RAM usage of this kernel is also much more stable. However, the quality of maps produced by Gmapping is significantly lower than LOAM, which is a critical trade off. Indeed, in the authors’ experience the maps produced by Gmapping are often insufficient for robot navigation, especially in large and/or complex environments. The summary of processes and threads of Gmapping in Table 3, (same interpretation as Table 2) also conveys the relative simplicity of Gmapping, being mostly a single process kernel as the other processes are just static transform broadcasters and contribute very little to the computational load. For the other two platforms, the behaviour and utilization of Gmapping SLAM is very similar and therefore we omit the detailed distribution plots for brevity. A fundamental insight is that the Gmapping is a much simplistic version of SLAM and also 2D. Therefore it is expected that it has very small variation across different platforms but prior to this study we have not found any findings that proves it with real world dataset. This finding is quite useful for users when planning for deployment and scheduling.

Next, in the Google cartographer SLAM, we have the justification for the assumption of memory consumption in 3D LOAM SLAM. Here we look into both 2D and 3D dataset provided by the developers in Deutsches Museum in the most resource constraint hardware, the Jetson Xavier NX. It is clearly evident how the system is overloading with memory consumption in the 3D version while having similar execution and utilization for the 3D and 2D versions. Investigating further into this issue, the cartographer 2D and 3D have similar structure as depicted in Fig. 4. They both produce local and global map outputs. However, the significant difference is the data that saves the map as they grow with the traversal in the environment. This growth in memory utilization is a severe concern for resource allocation as it will impede any
Fig. 7 Geographical locations of the data points for Fig. 8. Depicted is the execution times for LOAM in the Xavier AGX platform. The colors indicate the execution time required at a particular location (red is high, green is low).

other tasks running in the system parallel. This therefore is a significant finding of our experiments.

We depict in Figs. 12 and 13 the execution time, memory and CPU utilization of Cartographer SLAM in 2D environments and 3D environments respectively. The parameters have been tuned at their default value as set by the authors optimized for the Deutsches Museum dataset [39]. For the 3D Carto SLAM it is alarming to see the rise of memory utilization to almost the system capacity as it traverses the environment. It is mainly due to the 3D data also set at a higher range and resolution that the 2D data in the default tuning parameters by the authors.

However, we see that the Carto 2D has the utilization higher than the Carto 3D. This is mainly because the ceres scanmatcher [56] for the Global SLAM node for 2D Carto is set at a higher maximum iterations than the 2D Carto to do the optimizations. In both cases we see the sinusoidal patterns because the Global SLAM is running at a much slower rate than the local SLAM.

Finally we present the profiling results of ORBSLAM3, our only vision based SLAM kernel, in Fig. 14. The ORBSLAM3 generate sparse maps compared to the other three kernels but fast in execution as comparable to LOAM. The software system is complex in this kernel, making it unfeasible to use in severely resource constrained hardware or power settings. Therefore we show the results in AGX Xavier in maximum power mode. The dip in the execution times as well as the CPU utilization around 200 sampling point is
Fig. 8 Computational profile data of LOAM SLAM in Xavier AGX platform. a Execution time which includes User Time and System Times for an activation period of 0.1 sec. In the Right axis is RAM utilization as a percentage of used RAM to the total available RAM. b CPU utilization and power measurements. c Indoor and outdoor environment corresponding to data points circled in (a).
Fig. 9 Computational profile data of LOAM SLAM in Xavier NX. 
(a) Execution time which includes User Time and System Times for an activation period of 0.1 sec. 
(b) CPU Utilization and Power measurements.

Fig. 10 Computational profile data of LOAM SLAM in Zotac. 
(a) Execution time which includes User Time and System Times for an activation period of 0.1 sec. 
(b) CPU Utilization and Power measurements.
Table 2 Process/thread summary for LOAM in Xavier AGX platform

| Process               | Laser mapping | Laser odometry | Transform maintenance | Multi-scan registration |
|-----------------------|---------------|----------------|-----------------------|-------------------------|
| Number of threads     | 4             | 4              | 4                     | 4                       |
| % utilization of dominant thread | 98.97      | 97.72          | 60.7                  | 98.07                   |
| % of execution time   | 32.8          | 28.8           | 0.17                  | 38.13                   |
| Avg. execution time   | 0.08448       | 0.0736         | 0.0003                | 0.0947                  |
| Variance in Exec time | 0.00106719    | 0.0007106      | 2.91E-6               | 0.000108                |

Fig. 11 Computational profile data of Gmapping SLAM. a Execution time (User Time and System Times) for an activation period of 0.1 sec and corresponding RAM utilization. b CPU Utilization and Power measurements

contributed by the loss of tracking which is the first building block of ORBSLAM as depicted in Fig. 5. The regeneration of tracking after the loss is equivalent to building a new map on top of the older one. Therefore, we can see that the slight monotonic increase in execution time once we have consistent tracking function ongoing in the map. Just like the other 3D SLAMs discussed before, this one also shows an increase in RAM utilization, but its growth rate is smaller compared to Cartographer 3D, due to the sparse nature of the map data. The CPU utilization is however considerable due to the heaviness of the software blocks, which makes this a computationally expensive kernel that is only applicable for the high accuracy it generates [38].
Table 3  Process and thread’s profiling summary of SLAM Gmapping in Xavier AGX platform

| Process            | Gmapping main node | Static transform publisher | Base link-laser transform | Odom map broadcaster |
|--------------------|--------------------|-----------------------------|---------------------------|----------------------|
| Number of threads  | 4                  | 4                           | 4                         | 4                    |
| Number of threads  | 8                  | 4                           | 4                         | 4                    |
| % Util. of dominant thread | 32.82        | 87.87                        | 87.96                     | 88.27                |
| % of execution time| 97.5               | 0.8                         | 0.8                       | 0.7                  |
| Avg. execution time| 0.10231            | 0.00156                     | 0.0014                    | 0.00094              |
| Variance in exec time | 0.00385         | 0.15E-5                     | 1.4E-5                   | 9.5E-6               |

Fig. 12  Computational profile data of Google Cartographer 2D SLAM. **a** Execution time (User Time and System Times) measured at a frequency of 0.1 sec and corresponding RAM utilization. **b** CPU Utilization and power measurements.
To conclude, Fig. 15 presents the comparison of execution times for the two SLAM kernels over our three computational platforms. Here it is evident that the Google Cartographer is 4 times faster in execution time compared to the other SLAM kernels at the expense of CPU and memory utilization. This plot gives a valuable insight to the system designer with comparative analysis of the SOTA SLAM kernels. For example, we can see that Cart3D is the most desirable kernel with fastest map generation that is beneficial to safety, however it almost takes all the CPU resource. Therefore the ORB-SLAM3 can be a good alternative with similar performance and a little less CPU consumption for the Xavier AGX when we are trying to implement a system of multiple kernels with complex mission objectives.

RRT-based path planning

The RRT path planner was executed as a standalone python3 code using the libraries from [57]. As a standalone python code, only a single thread was monitored in all the platforms. For an RRT planner, as specified in Section III, step/sampling size (R), environment size, and obstacle density were varied for each of the computational platforms. The resulting computational profiles in Fig. 16 show the effect of these parameters on the execution times. Typically, the sampling or step size must be determined arbitrarily based on prior experience in related use cases. Generally, a lower sampling size will result in higher execution times but a larger sampling size can also generate infeasible paths as small environment features (such as obstacles) can be overlooked. Moreover, as
the density of obstacles and size of environment increase, the execution times also increase. A lower sampling size in a large area, such as a square kilometer, is not desirable to find paths in a timely manner. However, with the obstacle density increasing, higher sampling sizes also tend to generate infeasible paths. Thus, knowledge of the average and worst case execution times in such scenarios are a must to plan paths in a timely manner for autonomous robots in complex and/or dynamic environments. Obstacle Density is a crucial environmental parameter for path planning and its relation to response times are rarely studied in literature.

In Fig. 17, we demonstrate the effect of parallelizing the RRT kernel with the results derived from the CUDA enabled RRT [45]. However, to reap the benefits of parallel computation, we need to increase the obstacle density beyond 40%. This experiment is done in a 100 x 100 area with sampling size \( R = 1 \). Beyond 40% obstacle density, we observe a rapid decrease in response times, which is consistent with the results in [45].

**Task allocation**

The task allocation kernel is written in C code by the authors following the algorithms from the task allocation problem in [46]. As a standalone C code, this kernel was also monitored as a single thread in all the platforms. As stated in Section III, we vary the robot cardinality, \( N \), the requirement cardinality \( R \), and functionality cardinality \( F \) to generate a set of task allocation problems with varying difficulty. Specifically, our test cases are built by setting \( N \times \text{average}(F) = F \) and \( R = F/2 \). We choose \( N \) from 5 to 100. Figure 18 depicts the execution time variations due to these parameter changes. In addition, Fig. 19 validates the polynomial complexity of the kernel, originally proved in [46]. The useful insight for a single threaded kernel is quite straightforward from this data. The More power and frequency for each cores the platform has, the faster is completes the tasks. That’s why we see the execution times get lower from Nx to AGX to Zotac.

**Optical flow and monocular depth**

The CV kernels were not operated with a particular activation period but on a typical while loop with no timing constraints. Therefore, we also provide the turnaround time information for these kernels. Both kernels are also run standalone utilizing libraries from the Github repositories [5, 6] and take as input vision data from our real-world dataset. Figures 20 and 21 show the computation and power profiles for the monocular depth estimation kernel in maximum and low power modes respectively. The first insight is the high usage of the shared CPU/GPU memory on the Xavier platform, which is highly utilized and almost completely consumed for PwCNet. The high utilization of RAM is due to fact that memory is shared between the CPU and GPU via a SoC in the Xavier platforms and these kernels are using GPU intensive PyTorch libraries. Also there is a significant increase in the execution times and utilization of CPU in low power mode. The interesting aspect from Figs. 20 and 21 is that there are spikes of CPU usage in both of the power modes. This behavior is caused by the insertion of two out of context images in the dataset of our environment. The regular images that represent the steady parts of the graphs are from our own collected data in which the environment is gradually changing as the robot moves. However, when it faces an out-of-context or new environmental feature set, it spends a lot of time processing the change in the CPU. There is also evidence of a CPU bottleneck issue after analyzing this portion of execution in the NVIDIA Nsight system and thus there is an opportunity of optimizing the kernel for usage on embedded heterogeneous platforms like Xavier.

In Fig. 22, the profiles of fastdepthv2 are depicted. We can see a dramatic decrease in turnaround times due to the fast execution of this kernel utilizing the CUDA C library. This is in contrast to the PyTorch based MiDaS which is a slower as well as heavier library. The faster response comes at the cost of lower accuracy, which is measured to be 44% for fastdepthv2 in comparison to 84% in MiDaS. Moreover, we do not see the spikes in execution times due to abrupt change in scenes like MiDaS, again due to the lighter nature of this kernel.

The optical flow estimation kernel is conducted in a similar manner as the depth estimation kernel without the insertion of out-of-context images. It is fed with a set of sequential
images from the same path the SLAM kernels were conducted on. The profiling information is visualized in Figs. 23 and 24. Here, there is a slight variation in computational loads, which is attributed to changing scenarios in the dataset. There is a shift to CPU computation from GPU during those instances, which contributes to the increase in CPU execution time which is more evident in the low power mode. Moreover, the GPU resources are also utilized at a minimum level which is evident from the utilization summary of several GPU parameters. The achieved Occupancy for MiDaS and Pwc-Net are respectively 25.28% and 20.77% and all the functional unit utilization are low or idle (0-2) when analyzed using the Nvidia Profiler. The Warp Execution Efficiency for MiDaS and Pwc-Net were observed to be 96 and 95% whereas the Shared Utilization were low or mid level (0-6) for both cases. These data were observed for 61 and 35 kernel threads for MiDaS and Pwc-Net respectively and given as a summary or average here. For the Zotac platform, where we have to measure the GPU parameters using Nvidia-smi interface, we observe low utilization of the GPU resources. For Midas in Zotac, the GPU utilization is 35.5 with GPU memory utilization at 15.35775 % and a much high GPU power usage at 72 W. For Pwc-Net in Zotac, the GPU utilization is 16.4 %, memory utilization at 16.9 %, and GPU power usage at 46 W.

We depict the distribution of the different parameters over the dataset of both of these CV kernels in Figs. 20, 21, 23 and 24 for the Xavier AGX platform only. For the other platforms, we observe similar spikes in computation for the out-of-context images for depth estimation. For the optical flow estimation, the execution times are also very similar. However, the resource utilization for the Zotac is quite low compared to the Xaviers and the data as well and cause of this is discussed in the next section. We again omit the similar figures of distribution of execution time and resource utilization for Xavier NX and Zotac for brevity. Finally we present in Fig. 25 the boxplots for the computer vision kernels across our computational platforms to represent these comparisons.

It is visible that in the Zotac platform, the execution times are higher whereas the overall response times are proportionally lower. This is attributed to the fact that the higher CPU core count of Zotac yields more parallelism of the vision kernels in Zotac compared to the Xaviers and thus overcomes the CPU bottleneck issue in the Xaviers to some extent.

Remark 4 The data collection experimentation and also corresponding images related to SLAM and MiDaS are depicted in the video https://youtu.be/QQ8IeRK0PgE. This gives a clearer picture of the environment the experiments were conducted in, including both outdoor and indoor scenarios.

Discussions

Our profiling efforts yield several significant insights that we argue are fundamental to our goal of guaranteeing the timeliness and correctness of complex autonomous behaviors in robotic systems. First, from all the data presented earlier as well as the summary in Table 4 it is quite evident that the operation of these kernels simultaneously, which is necessary for complex robotic applications, will degrade performance and yield missed deadlines (which are equal to the activation periods), especially in low power scenarios. While the overall utilization is not merely a summation of the individual utilization, the surprising changes in response times in certain scenario can generate these deadline misses. A change in activation periods and appropriate scheduling as well as prior knowledge of these changes in response/execution times will be necessary to operate a combination of CV and CPU-bound kernels, which if done naively may yield catastrophic impacts on performance and safety of autonomous behaviors. This motivates our ongoing efforts to develop a dynamic scheduling parameter adjustment for autonomous robots based on
Fig. 16  Boxplots of execution or cpu user times of RRT path planning experiments

Fig. 17  Comparative analysis of boxplots of execution times of parallelized RRT path planning experiments for both CPU and GPU loads
the data presented in this work. Figure 26 refers to an example application using the computational kernels that can be scheduled in a Directed Acyclic Graph (DAG) scheduling framework such as implemented in ROSCH [58]. In any such DAG scheduler, the comprehensive knowledge of individual execution times are a must for designing a dynamic computationally aware scheduler [59, 60]. Most importantly, as we argue in the paper, our analyses of computational parameters are a prerequisite for any real-time scheduling methodology and also inform the design of appropriate schedulers for robotics kernels, as we are pursuing in our ongoing work. This is the pivotal part of the learning and regression in our proposed workflow (Fig. 27) for computational awareness in autonomous robots.

Finally we also derive additional insights into scheduling, optimization and resource management for future deployment of these kernels simultaneously.

**Remark 5** Related to the above, the importance of middleware during full system deployment is high. ROS is the most used middleware in research. However, for industry there are other choices such as Kafka and RabbitMQ. Moreover, with the advent of ROS2.0, there has been a wide range of research on evaluating the performance of ROS2.0 [61–63]. ROS2.0 also integrates not only QoS but also real time capabilities that were not present in ROS. With all these topics, we wanted our metrics to be neutral of any middleware and so we designed our experiments in such a way that the profiles are expected to hold true for any middleware. However the effect of middleware is surely a significant topic of future research.

**System design insights** All of these numbers give us an idea of what kind of hardware platform we should use for our robots. Mainly if we are looking for longer deployment, the Xavier AGX or NX are better suited due to its low power consumption. If we can supply with a high amount of battery power, the Zotac or Intel NUC is better suited with performance that can guarantee safer operation of robots.

In to deeper details, in Fig. 28, we depict the data in a qualitative way to represent which SLAM kernel is more suitable for a particular robotic deployment. From the literature, we know the quality of the mpa outputs for the individual SLAM kernels; however, this analysis gives us deeper understanding of system design requirements. For example, if we are
Concerned about power consumption, the LOAM 3D SLAM might be a better potion than the more accurate but more power-consuming cartographer.

Moreover, the GPU intensive vision kernels MiDaS and PWC-Net were almost consuming all the resource of NX, which makes it impractical to run these kernels without any optimization on NX, meaning aerial vehicles may not have access to such vision capabilities. We present such results so that a designer can make the crucial decision of choosing the kernels according to the available hardware capabilities.

In this paper our objective is to show certain trends and insights into robotic computational kernels that would hold true for hardware updates unless there is a drastic change, such as a new computational platform that accelerates a particular form of computation. Similarly for software updates we would expect the computation to get further improved, but in a platform like Xavier NX which is so constrained on computing resources and power, we still expect to see the issues faced by MiDaS and PWC-Net. Real-time systems are very specific to the hardware itself, which makes the analysis this paper more meaningful as we demonstrate a process of software evaluation that is necessary for any system deployment.

**Environmental correlation** Next, the LOAM kernel shows a spatial correlation in the event of loop closure and revisiting environmental cues with a drop of the execution time and other related parameters. However, the memory usage continues growing with the traversal of the environment. In about 30 min of traversal, the RAM usage grows from 6% to 12% in the Xavier AGX platform, with similar growth in Xavier NX and Zotac. This trend is seen to be present in both 2D and 3D Cartographer SLAM, but mostly prevalent in 3D cartographer. The range and resolution of 3D grids seem to have a very significant impact on memory consumption. The Cartographer, however known for its speed and accuracy, is much faster than the other SLAM kernels proving its applicability in high capacity systems as the CPU and memory utilization are almost consuming the whole system for the resource constrained Xavier NX platform. On the other hand, the Gmapping SLAM kernel shows a quite stable behavior, but with significantly lower map quality. Interestingly, in low power mode the LOAM kernel is inoperable at the default activation period of 0.1 s as 100% utilization is seen, resulting in missed execution deadlines. These SLAM insights allow for unique opportunities such as dynamic scheduling parameter adjustments that anticipate loop closure events, or robot mission planners that balance location revisitation with the availability of computational resources.

**Remark 6** The environmental factors that can affect the computation are nature of data (e.g., point cloud or vision-based), environment type such as indoor vs. outdoor, features of environment (such as dynamism), etc. Moreover, the different hardware platforms also play a vital role in computation. In our tests, we have used similar input dataset for all the pointcloud based SLAM (same input dataset for LOAM and Gmapping) at a constant hardware setting. This ensures that the external factors that influence computation are fixed and the nature of the method itself will dictate computational performance. This reflects our best effort to make all analysis fair in the testbench. Given this setup we believe that with some margin of error, a similar dataset will yield similar results as we observed in our tests, which for SLAM has very clear relationships between the robot’s deployment environment and the computational load of SLAM.

| Platform | Kernels | Power mode | CPU util (%) | RAM util (%) |
|----------|---------|------------|--------------|--------------|
| LOAM 3D SLAM | Max | 57 | 10–22 |
| Carto 3D SLAM | Max | 63 | 10–90 |
| Carto 2D SLAM | Max | 72 | 1–18 |
| Nvidia | Gmapping 2D SLAM | Max | 31 | 32 |
| Xavier | RRT path planning | Max | 18 | 24.5 |
| NX | Task allocation | Max | 20 | 23.5 |
| MiDaS depth est | Max | 10.2 | 91 |
| Pwc-Net optical flow est | Max | 19 | 89.4 |
| ORBSLAM3 | Max | 45 | 5–12 |
| LOAM SLAM | Max | 32 | 5–20 |
| Nvidia | Gmapping SLAM | Max | 22 | 12 |
| Xavier | RRT path planning | Max | 13.2 | 17.1 |
| AGX | Task allocation | Max | 14 | 17.7 |
| Fastdepthv2 | Max | 12 | 20 |
| MiDaS depth est | Max | 12.2 | 56 |
| 15W | 26 | 55 |
| Pwc-net optical flow est | Max | 13.4 | 86.7 |
| 15W | 27.8 | 40 |
| LOAM SLAM | Max | 18.7 | 6–11 |
| Gmapping SLAM | Max | 15.3 | 8 |
| Zotac | RRT path planning | Max | 8 | 4.5 |
| Task allocation | Max | 8.3 | 8.4 |
| MiDaS depth est | Max | 27.2 | 12.1 |
| Pwc-net optical flow est | Max | 6.9 | 13.5 |

Shifting to the task allocation and RRT path planning kernels, we see an even greater variation in both average and WCET as evident from the boxplots in Figs. 16 and 18, with these effects being directly dependent on a robot’s operational environment and/or objectives. Once again such insights open up dynamic resource management opportunities, such as controlling the quality of planned paths based on computational resources and safety margins, or mission planners that anticipate task loads in an environment and act...
to mitigate impacts on kernel scheduling. The single threaded processes such as RRT and TA can have differences in execution times between the AGX and NX platforms because the NX is more conservative in power consumption than the AGX. This can affect the execution of kernels specially for higher periods of executions. That is evident in Fig. 16. Moreover, we can utilize the GPU for optimization of RRT and TA by running the critical sections on the GPU as seen in Fig. 17.

**CV kernels and CPU-GPU interaction** Focusing now on the CV kernels, significant variation in resource utilization is observed with changing Xavier power modes. For example, the execution time increases 3 fold both in the Midas depth estimation and optical flow estimation. The vision kernels also show a classic example of GPU starvation and CPU
bottleneck in Xavier hardware. Parts of the algorithm are sequential with no parallel operation implemented, in particular, portions dedicated to preprocessing the images before calling the PyTorch libraries. Most of the GPU resources are underutilized while occupying a huge amount of shared memory space. This usage of a huge amount of memory calls for a conflict resolution in CPU and GPU memory access, which is investigated in very recent literature [31–33] and can be put into action using the data from this paper. In the Zotac system, which has a discrete GPU unlike the Xaviers, the CV kernels show much better performance and lower CPU bottleneck issue, however the Zotac platform is not viable for aerial robots. This final point indicates that for vision kernels there may be a strong case for offloading computation to more capable agents (e.g., from an aerial robot to a ground robot or the cloud).

Although it bridges the gap between theoretical and practical usage of robotic computational kernels, some of the results in this paper are quite predictable. However, the Pwc-Net results in Fig. 25 depicts quite the unpredictable behaviour. It reflects that only using powerful hardware with discrete GPU cannot yield better response times for all the pytorch enabled computation and some DNN implementations are more compatible to the embedded hardware like Xaviers than the others. It however uses much less RAM in Zotac due to its discrete GPU memory.

Summary of insights A final insight from the summary in Table 4 is that the memory conflict for concurrent operation of the kernels may also influence the schedulability and predictability of operation similar to the typical CPU resource management schemes.

(a) GPU-intensive kernels: The Deep Neural Network (DNN) based algorithms have various ways of optimization depending on the hardware such as utilizing the vision accelerators in the Xaviers as well as generalized approximation techniques
such as matrix factorization, compression. Moreover, GPU intensive kernels also have environmental correlation which can be modeled and further studied with empirical data such as the other kernels in this paper.

(b) CPU-intensive kernels: The analysis in this paper utilizes the default Completely Fair Scheduler (CFS) of the Linux kernel. Specifically, the parameters which could be dynamically adjusted for scheduling these computational kernels via Linux schedulers are priorities, scheduler tick, vruntime, deadlines and the scheduling mechanism itself. Therefore, our followup work is on analyzing real time schedulers such as FIFO/RR/EDF and in that case we will also define hyperperiod of tasksets and target latencies and also analyse the effects of these schedulers on the same parameters as this paper but with a combinatorial implementation of the computational kernels. Such experiments will generate a learned computational map which can be used for our ultimate objective of computation aware autonomy, as specified in Fig. 27. There are several scheduling algorithms that can be utilized such as deadline based hard real time, soft real time and novel environment aware schedulers. Usage of RT(Real-Time)-patches and RTOS(Real Time Operating System), CPU isolation, memory locking are also some of
the possible options. The applicability of these will depend heavily on the mission objectives.

(c) Combining a and b: Moreover, the Jetson Xavier architecture shares the RAM between the CPU and GPU, therefore the GPU intensive kernels have a much higher RAM utilization. This could pose an issue when used combinatorially with other kernels and some literature which address these memory access issues in heterogeneous architectures are cited in the discussions [31–33].

**Remark 7** An important note is the presence of noise in our data, stemming primarily from our data sampling mechanism and file writing operations for profiling result collection. The use of an average differential mechanism keeps the variance of that noise present in data. However, we have also profiled the noise and account for its average value. Specifically, we can quantify that the error margin of our presented data is 2.36% for CPU utilization, 0.015% for RAM utilization and 6.9% for CPU power. The execution times are devoid of this error.

**Conclusion**

Autonomous systems cannot treat computation as a black box anymore as has been done in most of the systems deployed in present time. Our study provides a detailed preliminary insight into the means of achieving the timeliness guarantee. This paper aims to gain deeper insights into the robotics computational kernels in real environments. To limit the experiments’ extent to meaningful and interpretable content, we choose a set of representative computational kernels needed for broader autonomy. Most of them are from existing libraries except the Task Allocation kernel (which was
developed by the authors due to unavailability of C code implementation). We rigorously analyze the effects of environmental parameters such as depth and motion features, complexities of indoor/outdoor, size and obstacle densities. While the results were mostly not surprising, our quantified profiles fill the gap of current literature for timely and safe autonomy. The work on environmental correlation of SLAM and RRT path planning kernels are of vital importance for practical applications. Moreover, for the application of a DAG scheduling comprised of these computational kernels, this individual execution times measurements are considered as input data for the schedulers. The experimental real-world dataset along with the profiling results are the necessary first step towards future research on computational awareness on autonomous robots. We also use existing and well renowned tools such as psutil so that the methodologies are not questionable. We have not changed the many different features on the hardware platforms, but still from the results, it can be deduced that the number of cores, maximum power consumption, operating frequency and also architecture difference such as integrated or discrete GPU all have an effect on the performance.

**Funding** National Science Foundation CNS-1932074.

**Data availability** Dataset are given as download links in the code’s github repository here: https://github.com/caslab-vt/CAA-profile.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

### References

1. Ermolov I (2020) Industrial Robotics Review, pp. 195–204. https://doi.org/10.1007/978-3-030-37841-7_16
2. Grisetti G, Stachniss C, Burgard W (2007) Improved techniques for grid mapping with ra-o-blackwellized particle filters. IEEE Trans Robotics 23(1):34–46. https://doi.org/10.1109/TRO.2006.889486
3. Marder-Eppstein E, Berger E, Foote T, Gerkey B, Konolige K (2010) The office marathon: Robust navigation in an indoor office environment. In: 2010 IEEE International Conference on Robotics and Automation, pp. 300–307
4. Liu J, Williams RK (2019) Submodular optimization for coupled task allocation and intermittent deployment problems. IEEE Robot Autom Lett 4(4):3169–3176. https://doi.org/10.1109/LRA.2019.2925301
5. Sun D, Yang X, Liu M.-Y, Kautz J (2018) Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In:Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8934–8943
6. Ranftl R, Lasinger K, Hafner D, Schindler K, Koltun V (2019) Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. arXiv:1907.01341
7. Guclu O, Can AB (2019) Fast and effective loop closure detection to improve slam performance. J Intell Robot Syst 93(3–4):495–517
8. Belavadi S.S, Beri R, Malik V (2017) Frontier exploration technique for 3d autonomous slam using k-means based divisive clustering. 2017 Asia Modelling Symposium (AMS), 95–100
9. Noreen I, Khan A, Ryu H, Doh N, Habib Z (2017) Optimal path planning in cluttered environment using rtt*-ab. Intelligent Service Robotics 11. https://doi.org/10.1007/s11370-017-0236-7
10. Nvidia: Jetson AGX Xavier. https://devblogs.nvidia.com/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/. Accessed: 2020-01-30
11. Wei H, Shao Z, Huang Z, Chen R, Gnan Y, Tan J, Shao Z (2016) Rtos: a real-time ros architecture on multi-core processors. Future Gener Comput Syst 56:171–178. https://doi.org/10.1016/j.future.2015.05.008
12. Delgado R, You B-J, Choi BW (2019) Real-time control architecture based on xenomai using ros packages for a service robot. J Syst Softw 151:8–19. https://doi.org/10.1016/j.jss.2019.01.052
13. Maruyama Y, Kato S, Azumi T (2016) Exploring the performance of ros2. In: Proceedings of the 13th International Conference on Embedded Software. EMSOFT-16. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2968478.2968502
14. Wang Y, Liu S, Wu X, Shi W (2018) Cavbench: A benchmark suite for connected and autonomous vehicles. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 30–42. https://doi.org/10.1109/SEC.2018.00010
15. Clemons J, Zhu H, Savarese S, Austin T (2011) Mevbench: A mobile computer vision benchmarking suite. In: 2011 IEEE International Symposium on Workload Characterization (IISWC), pp. 91–102. https://doi.org/10.1109/IISWC.2011.6114206
16. Kato S, Tokunaga S, Maruyama Y, Maeda S, Hirabayashi M, Kitsukawa Y, Monrroy A, Ando T, Fujii Y, Azumi T (2018) Autoware on board: Enabling autonomous vehicles with embedded systems. In: 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPPS), pp. 287–296. IEEE
17. Autoware: Open-source software for urban autonomous driving. https://github.com/Autoware-AI/autoware.ai
18. Zhao H, Zhang Y, Meng P, Shi H, Li L, Lou T, Zhao J (2019) Towards safety-aware computing system design in autonomous vehicles. arXiv:1905.08453
19. Carlone L, Karaman S (2018) Attention and anticipation in fast visual-inertial navigation. IEEE Trans Robot. https://doi.org/10.1109/TRO.2018.2872402
61. Maruyama Y, Kato S, Azumi T (2016) Exploring the performance of ROS2. Proceedings of the 13th International Conference on Embedded Software, EMSOFT 2016, 0–9. https://doi.org/10.1145/2968478.2968502

62. Blass T, Hamann A, Lange R, Ziegenbein D. Automatic Latency Management for ROS 2: Benefits, Challenges, and Open Problems, 1–14

63. Fernandez J, Allen B, Thulasiraman P, Bingham B (2020) Performance study of the robot operating system 2 with QoS and cyber security settings. SYSCON 2020- 14th Annual IEEE International Systems Conference, Proceedings. https://doi.org/10.1109/SysCon47679.2020.9275872

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.