# A Safety-Performance Metric Enabling Computational Awareness in Autonomous Robots

Ashrarul H. Sifat, Xuanliang Deng, Burhanuddin Bharmal, Sen Wang, *Graduate Student Member, IEEE*, Shaoyu Huang, Jiabin Huang, Changhee Jung, *Senior Member, IEEE*, Haibo Zeng, and Ryan Williams, *Member, IEEE*

***Abstract*—This letter takes a first step towards the analysis of safety *and* performance critical computational tasks for autonomous robots. Our contribution is a safety-performance (SP) metric that ensures safety first and then rewards improved performance of real-time computational tasks, building on the notion of "nominal safety" which defines timely computation as critical to safety. To fully utilize the computing capacity of heterogeneous processing units (e.g., CPU + GPU), a computational task graph model called the Stochastic Heterogeneous Parallel Directed Acyclic Graph (SHP-DAG) is adopted to capture the uncertain nature of robotic applications and their required computation. Compared to state-of-the-art task models, SHP-DAG avoids the pessimism of deterministic worst-case execution time (WCET), instead modeling the execution times of tasks by probability distributions. Our SP metric is defined upon this task model, which allows us to apply the FIFO and CFS schedulers of the Linux kernel on complex robotic computational tasks and compare the SP metric with baseline metrics, average and worst-case makespan. Extensive experimental results on NVIDIA Jetson AGX Xavier hardware demonstrate that the proposed SP metric is appropriate for managing computational tasks in a manner that balances safety and performance in robotic systems.**

***Index Terms*—Autonomous robots, safety management, scheduling algorithms, software performance, uncertainty.**

## I. Introduction

AUTONOMOUS robots often have a sophisticated set of objectives, e.g., as seen in multi-robot search and rescue and precision agriculture [1], [2], and face difficult restrictions on computation while operating in dynamic environments. Additionally, the computational tasks that support autonomy may vary widely in their impact on safety (based on task timeliness), while others vary in their impact on performance (based on quality of task output). To guarantee robot safety while maintaining performance over broader mission objectives, the computation that supports autonomy must be managed to balance between safety and performance [3].

We have seen related thrusts in the autonomous vehicle (AV) domain [4], [5], [6], which work with the concept of "nominal safety" based solely on task responsiveness for collision avoidance [6], [7]. However, we argue that in the broader context of robotics, the notion of nominal safety and related metrics must take on a more complex form. Thus, this work aims to create a metric that *balances safety and performance* for complex, nondeterministic robotic pipelines. Comparing to the "safety score" for AVs [6], our metric captures variation in computation and can trade off safety-driven computation with performance-driven computation once the system is safe.

In defining our metric, we adopt the *Stochastic Heterogeneous Parallel Directed Acyclic Graph (SHP-DAG)* model, where task response times are expressed as probability distributions to capture the uncertain nature of a robot's computation (Section II). Based on this computational model, we then derive our safety-performance (SP) metric that ensures safety first and then rewards improved performance of real-time computational tasks (Section III). Finally, we perform extensive experiments on NVIDIA Jetson AGX Xavier hardware with real-time Linux schedulers (Section IV), and demonstrate that our SP metric is appropriate for managing computational tasks in a manner that balances safety and performance.

*Related Work:* Safety-focused frameworks have been studied extensively for collaborative robots carrying out tasks in confined spaces with human involvement. For example, safety assessment methods based on kinetostatic safety fields [8], safety-driven robot application design methods [9], automating traditional risk analysis methods [10], as well as fuzzy logic system (FLS) and reinforcement learning (RL) based comparative analysis [11] have been proposed to prevent collisions in collaborative robot workspaces. While such works have a clear safety objective, they often do not treat computation explicitly, or have computational tasks that are relatively simple.

In contrast, tremendous work has been performed in the real-time computing community, where Directed Acyclic Graphs (DAGs) have become a popular method of modeling

Manuscript received 10 March 2023; accepted 30 June 2023. Date of publication 31 July 2023; date of current version 7 August 2023. This letter was recommended for publication by Associate Editor C. I. Vasile and Editor L. Pallottino upon evaluation of the reviewers' comments. This work was supported by NSF under Grant CNS-1932074. *(Ashrarul H. Sifat and Xuanliang Deng contributed equally to this work.) (Corresponding author: Ashrarul H. Sifat.)*

The authors are with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA 24061 USA, also with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA, and also with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA (e-mail: ashrar7@vt.edu; xuanliang@vt.edu; burhanuddinb@vt.edu; swang666@vt.edu; huan1464@purdue.edu; jbhuang@umd.edu; chjung@purdue.edu; hbzeng@vt.edu; rywilli1@vt.edu).

Digital Object Identifier 10.1109/LRA.2023.3300251

computation. While such results are powerful analytically, there is a gap at the intersection with robotics. A relevant example of work at this intersection is smARTflight [12], which proposes an environmentally-aware unmanned aerial vehicle (UAV) flight management system that adapts the execution frequencies of flight control tasks according to timing and safety-critical constraints. However, [12] applies deterministic metrics for guiding computational resource management, and focuses only on the task of flight control. Finally, there are recent scheduler implementations based on DAGs such as a fixed-priority based DAG scheduler utilizing the Robot Operating System (ROS) [13], and a real-time scheduler and analysis of processing chains in ROS2 [14]. While these methods are exemplary developments in managing computation on robots with complex DAG models, they lack a unified framework that can ensure safety and performance under uncertainty.

Finally, there has been work in the high performance computing (HPC) domain to capture probabilistic execution times instead of deterministic worst cases [15], [16]. While we take inspiration from these works, they do not consider safety-performance trade-offs, nor do they characterize the computation for robotic systems. To our knowledge, our work is the first to consider probabilistic timing constraints with formalism for achieving safety-performance trade-offs.

## II. Problem Formulation

### A. Computational Model

We start by defining concepts from real-time computing (see Fig. 1). In this letter, we consider a *computational task* as an algorithm in support of autonomy (e.g., planning, vision, etc.) that executes on an embedded processor.

*Definition II.1 (Start Time):* The *start time (ST)* for a computational task is the timestamp when task execution begins.

*Definition II.2 (Execution Time):* The *execution time (ET)* for a computational task is the amount of time the task spends in a processor of the host operating system.

*Definition II.3 (Response Time):* The *response time (RT)* for a computational task is the amount of time taken to generate output from input for a particular task. In general,

$$RT = ET + IT \tag{1}$$

where $IT$ is the *idle time* spent waiting for resources.

*Definition II.4 (Activation Period):* The *activation period (AP)* for a computational task is the amount of time between two consecutive executions of the task.

*Definition II.5 (Timing Distribution):* Multiple executions of a task $i$ yields a set of $m$ measurements $\mathcal{M}_i$ containing $RT$ and $ET$ for each execution. *Timing distributions* can then be built based on $\mathcal{M}_i$, denoted by symbols $R$ and $X$, that capture the variation in RT and ET of task $i$ (see Section III).

The above definitions now allow us to define our concepts of safety and performance for robot computation.

*Definition II.6 (Safety):* The *safety* of computation is based on *probabilistic timeliness* of computation, i.e., $P(R > \tau) \leq \lambda$,

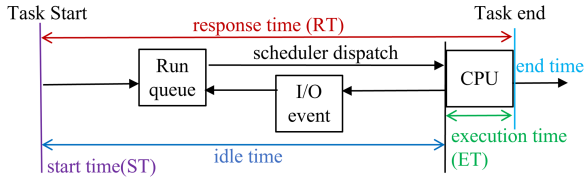


Fig. 1. Diagram of task timing parameters. Note that RT and ET are a time duration, whereas ST and ET are time instants.

where $R$ is a random variable described by an RT distribution, $\tau$ is a nominally safe timing, and $\lambda$ is a probabilistic threshold.

*Definition II.7 (Performance):* The *performance* of computation is based on the share of computational resources received from the host operating system relative to ET distribution $X$.

Intuitively, we consider safety as derived from timely output from computation in safety critical scenarios (e.g., for collision avoidance), and performance as derived from larger shares of computational resources allowing for better solutions to hard problems (e.g., planning). With the above definitions, we now formalize the computational model adopted in this work, the *Stochastic Heterogeneous Parallel DAG (SHP-DAG)*.

*Definition II.8 (SHP-DAG):* A real-time robotic application composed of computational tasks is represented by an *SHP-DAG*, a *directed acyclic graph* defined as $G = (\mathcal{V}, \mathcal{E}, Type, Tag, \mathcal{R}, \mathcal{X}, \mathcal{A}, A_{DAG})$:

- $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the set of IDs for all $n$ graph nodes. Each node $v_i \in \mathcal{V}$ is a computational task that executes on a specific type of processor.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of *directed edges* among tasks that indicates the data flow from one task to another.
- $Type = \{type_{v_1}, type_{v_2}, \ldots, type_{v_n}\}$ is the set of types of all tasks. A node in an *SHP-DAG* has one of the following types in this work: {Computing, Data/Sensor}.
- $Tag = \{tag_{v_1}, tag_{v_2}, \ldots, tag_{v_n}\}$ indicates the processing unit that each task should run on (e.g., CPU or GPU).
- $\mathcal{R} = \{R_{v_1}, R_{v_2}, \ldots, R_{v_n}\}$ is the set of probability distributions of response times for all tasks.
- $\mathcal{X} = \{X_{v_1}, X_{v_2}, \ldots, X_{v_n}\}$ is the set of probability distributions of execution times for all tasks.
- $\mathcal{A} = \{A_{v_1}, A_{v_2}, \ldots, A_{v_n}\}$ is the set of task APs.
- $A_{DAG}$ is the overall activation period of $G$, which defines the rate at which $G$ is re-executed.

An illustration of the SHP-DAG model is given in Fig. 2, which captures the case study in this work (full details in Section IV). From the SHP-DAG definition and Fig. 2, we see that robot computation is modeled as a sequence of computational tasks that operate on input data (from sensors, or output from other tasks) and produce an output for subsequent tasks. Mathematically, such a sequence is a *path* in $G$, where we denote by $\mathcal{P}$ the set of all paths in $G$. To capture safety in such a model, consider the following definitions.

*Definition II.9 (Safety-Critical Path):* Given an SHP-DAG $G$, a *safety-critical path* $\ell_i \in \mathcal{C}$ is a chain nodes in $\mathcal{V}$ that achieve functionality that is critical to the safety of the application, where $\mathcal{C} \subseteq \mathcal{P}$ is the set of all safety-critical paths in $G$.

*Definition II.10 (Safety-Critical Node):* Given an SHP-DAG $G$, a *safety-critical node* $v_i \in \mathcal{V}_c$ is a computational task that

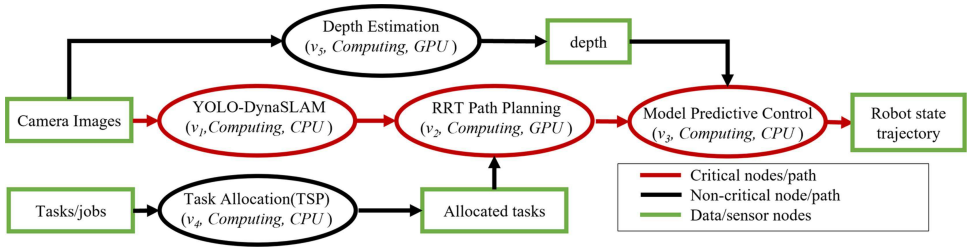


Fig. 2. Example of our *SHP-DAG* model, shown with computational tasks and task dependencies seen in mobile robots. We denote the critical paths and nodes by red ovals and arrows respectively, where we have one critical path and three critical nodes.

achieves a functionality that is critical to the safety of the application, where $\mathcal{V}_c \subseteq \mathcal{V}$.

We model specific terms for critical nodes as there may be instances where a timing constraint for a critical path is satisfied but the system remains unsafe. For example, if a localization and mapping node is too slow, even if the computational path it lies on meets a timing constraint, the staleness of the map may endanger the system or bystanders.

The above concepts allow a system designer to designate paths and nodes in an SHP-DAG as critical for safety (as in Fig. 2), which will then be captured in our SP metric as upper bounds on response times (Section III), yielding rewards for responsive computation. All non-critical paths $\ell_i \in \mathcal{P} \setminus \mathcal{C}$ and nodes $v_i \in \mathcal{V} \setminus \mathcal{V}_c$ will then be evaluated for their contribution to performance based on execution time.

### B. DAG Paths and Makespan

As outlined above, a DAG path[1] is a sequence of communicating tasks where every task receives data from its predecessor. In the real-time computing literature, it is most common to evaluate DAG paths through *Makespan*, i.e., end-to-end latency [17]. Makespan of a DAG path is the length of time that elapses from the start of the path to the end. As we use Makespan as a baseline for comparison with our SP metric (Section IV), consider the following definition:

*Definition II.11 (Makespan):* The *Makespan* of a path $\ell$ in an SHP-DAG $G$ is given by:

$$RT_\ell = (ST_{v_{\text{end}}} - ST_{v_{\text{start}}}) + RT_{v_{\text{end}}} \tag{2}$$

where $v_{\text{start}}$, $v_{\text{end}}$ are the start and end nodes of path $\ell$.

We see that the Makespan of a path is essentially the path response time. Finally, we also consider the worst case Makespan (WCMS) and average Makespan (AMS) for our comparison with the SP metric, based on the definitions below.

*Definition II.12 (WCMS & AMS):* The WCMS of a DAG path is the largest Makespan among all of its executions within a window of measurement whereas the AMS is the average of it. In our experiments that window is fixed, $A_{DAG}$, and the number of executions are variable based on the instantaneous Makespans. Considering $m$ executions of a DAG path $\ell$ within the period $A_{DAG}$, WCMS and AMS are defined by,

$$WCMS_\ell = \max_m RT_\ell \tag{3}$$

[1]Note, in the real-time community this is also known as a *task chain*.

TABLE I
SCHEDULING CLASSES AND POLICIES IN THE LINUX KERNEL

| Classes | Policies | Characteristics |
|---|---|---|
| Stop | No policy | Highest Priority |
| Deadline | SCHED_DEADLINE | Periodic hard real time tasks |
| Real Time | SCHED_FIFO,<br>SCHED_RR | Task priorities: 0 - 99<br>short latency sensitive, soft-real time |
| Fair | SCHED_OTHER,<br>SCHED_BATCH,<br>SCHED_IDLE | Completely Fair Scheduler (CFS)<br>Default scheduler in Linux<br>Motivated by Rotating Staircase Deadline |
| Idle | No policy | Lowest priority |

$$AMS_\ell = \frac{1}{m}\sum_m RT_\ell \tag{4}$$

### C. Real-Time Systems and Scheduling

Our concepts above are a basis for modeling a *real-time system*, a system that provides guaranteed response times for events and transactions. *Scheduling algorithms* are the key mechanism for achieving such real-time behavior as they balance computational resources among tasks, usually according to some metric (such as Makespan or our own SP metric). As ROS is primarily operated in Linux, we provide here a brief summary of the Linux schedulers, summarized in Table I.

In this letter, we utilize the real-time FIFO scheduler which operates according to fixed task priorities, and the default CFS scheduler which cannot provide real-time behavior. We design a study of our SP metric (Section III) around a combinatorial set of FIFO priorities (higher value is higher priority), with CFS as a baseline (Section IV). This study will illustrate that our SP metric is highly sensitive to FIFO priority for computation modeled as an SHP-DAG. Such sensitivity then paves the way for a *custom scheduler* based on our SP metric, i.e., solving the following problem:

*Problem 1:* A general scheduling problem optimizing the SP metric is given by:

$$\begin{aligned} \max_{\mathcal{S}} \quad & SP(\mathcal{S}) \\ \text{s.t.} \quad & schedulability(\mathcal{S}) = 1 \end{aligned} \tag{5}$$

where $\mathcal{S}$ is scheduler parameters that influence RT and ET.

With this problem context, our work focuses on developing $SP(\mathcal{S})$ and demonstrating the metric experimentally with $\mathcal{S}$ containing Linux scheduler parameters. However, we also note important features of the constraint $schedulability(\mathcal{S}) = 1$. In our formulation, this constraint would be defined as all probabilistic timing thresholds $\tau$ being satisfied, guaranteeing safety.

The probabilistic nature means that our problem would fall under the class of *black box methods*, where schedulability is verified by simulation as opposed to mathematical proof. In this case, methods proposed in our work [18] can be adapted to solve the optimization problem 1 (our future work).

## III. Safety-Performance Metric

### A. Response and Execution Times

To begin, we must populate the sets $\mathcal{R}, \mathcal{X}$ in an SHP-DAG $G$ that define task response and execution times, respectively. Most importantly, as opposed to the typical use of worst cases to define response/execution times, we aim to define *distributions*. Thus, we propose to measure response and execution times for the tasks $v_i$ and paths $\ell_i$ in $G$, and then use this data to build distributions.[2] Illustrating this process for task response times, consider a set of measurements $\mathcal{M}_i = \{RT_{i,j} \mid j \in [1:m]\}$, with $m$ measurements. Given that we have $\mathcal{M}_i$, we can define a *Response Time Profile (RTP)* $\mathcal{R}_i$ for task $v_i$, a continuous random variable defined on response time values $RT_{i,j}$. Thus, we can define the Probability Density Function (PDF), $pdf_{\mathcal{R}_i}$,

$$pdf_{\mathcal{R}_i}(RT_{i,j}) = P(\mathcal{R}_i = RT_{i,j}) \tag{6}$$

Now, the density function can be built assuming the data comes from a known distribution, such as Gaussian, Weibull, piecewise Normal, etc [15], [20]. For instance, a Gaussian distribution (which we use in our experiments) can be defined based on mean and standard deviation (SD) calculated from $\mathcal{M}_i$. Specifically, if the mean $\mu_i$ and SD $\sigma_i$ are calculated based on $\mathcal{M}_i$, then $pdf_{\mathcal{R}_i}(RT_{i,j})$ is given by

$$pdf_{\mathcal{R}_i}(RT_{i,j}) = \left(2\pi\sigma_i^2\right)^{-0.5} \mathrm{e}^{-(RT_{k,i}-\mu_i)^2/\left(2\sigma_i^2\right)} \tag{7}$$

Response time distributions for paths, and execution time distributions in $\mathcal{X}$ are built analogously.

### B. Definition of Penalty and Reward Functions

With our SHP-DAG $G$ fully defined, we can now build our SP metric by first considering that our metric should penalize when safety-critical paths and/or critical nodes violate timing constraints based on a particular schedule (i.e., $\mathcal{S}$). We begin with the first term of our metric which penalizes unsafe critical paths.

*Definition III.1 (Critical Path Penalty):* The penalty for safety-critical paths that violate timing constraints is:

$$f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S}) = \sum_{\ell_i \in \mathcal{C}_{\mathrm{us}}} p^{\ell_i}_{\mathrm{cp}}(P(R_{\ell_i} > \tau_{\ell_i}) - \lambda_{\ell_i}) \tag{8}$$

where $\mathcal{C}_{\mathrm{us}}$ is the set of safety-critical DAG paths that violate a *probabilistic* timing constraint, that is, $P(R_{\ell_i} > \tau_{\ell_i}) > \lambda_{\ell_i}$ where $R_{\ell_i}$ is the random variable describing the uncertain response time of critical path $\ell_i$, $\tau_{\ell_i}$ is the *nominally safe* response time for path $\ell_i$, and $\lambda_{\ell_i}$ is the probabilistic timing constraint for $\ell_i$.[3] With these definitions, and noting that $f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S})$ represents a penalty term (p) for critical path violations (cp) based on schedule $\mathcal{S}$ with a generic penalty function $p^{\ell_i}_{\mathrm{cp}}(\cdot)$ for each critical path $\ell_i$, (8) can be interpreted as penalizing based on the *deviation* of every violating critical path from its probabilistic timing constraint. Thus, if there are no safety-critical paths that violate timing constraints based on $\mathcal{S}$ then $\mathcal{C}_{\mathrm{us}} = \emptyset$ and $f^{\mathrm{p}}_{\mathrm{cp}}(\mathcal{S}) = 0$ yielding no penalty.

*Definition III.2 (Critical Node Penalty):* The penalty for safety-critical nodes that violate timing constraints is:

$$f^{\mathrm{p}}_{\mathrm{cn}}(\mathcal{S}) = \sum_{v_i \in \mathcal{V}_{\mathrm{us}}} p^{v_i}_{\mathrm{cn}}(P(R_{v_i} > \tau_{v_i}) - \lambda_{v_i}) \tag{9}$$

where $\mathcal{V}_{\mathrm{us}}$ is the set of safety-critical DAG nodes that violate a *probabilistic* timing constraint, that is, $P(R_{v_i} > \tau_{v_i}) > \lambda_{v_i}$ where $R_{v_i}$ is the random variable describing the uncertain response time of critical node $v_i$, $\tau_{v_i}$ is the response time for node $v_i$ that ensures nominal safety, and $\lambda_{v_i}$ is the probabilistic timing constraint for node $v_i$. With these definitions, and noting that $f^{\mathrm{p}}_{\mathrm{cn}}(\mathcal{S})$ represents a penalty term (p) for critical node violations (cn) based on schedule $\mathcal{S}$ with a generic penalty function $p^{v_i}_{\mathrm{cn}}(\cdot)$ for each critical node $v_i$, (9) can be interpreted as penalizing based on the *deviation* of every violating critical node from its probabilistic timing constraint. Thus, if there are no safety-critical nodes that violate their timing constraints based on schedule $\mathcal{S}$ then $\mathcal{V}_{\mathrm{us}} = \emptyset$ and $f^{\mathrm{p}}_{\mathrm{cn}}(\mathcal{S}) = 0$ yielding no penalty.

With the penalties for our metric defined, we now describe rewards gained after timing constraints for safety-critical paths are satisfied. Critically, the following reward terms are non-zero *only* when there exists no critical path or node constraints that are violated, i.e., $P(R_{\ell_i} > \tau_{\ell_i}) \leq \lambda_{\ell_i}, \forall\, \ell_i \in \mathcal{C}$ and $P(R_{v_i} > \tau_{v_i}) \leq \lambda_{v_i}, \forall\, v_i \in \mathcal{V}_c$.

*Definition III.3 (Path Reward):* The reward for DAG paths, assuming safety-critical timing constraints are satisfied, is:

$$f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S}) = \Bigg[\sum_{\ell_i \in \mathcal{C}} r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\lambda_{\ell_i} - P(R_{\ell_i} > \tau_{\ell_i})) + r^{\mathrm{p},\ell_i}_{\mathrm{path}}(X_{\ell_i})\Bigg] + \sum_{\ell_i \notin \mathcal{C}} r^{\mathrm{p},\ell_i}_{\mathrm{path}}(X_{\ell_i}) \tag{10}$$

*Definition III.4 (Node Reward):* The reward for DAG nodes, assuming safety-critical timing constraints are satisfied, is:

$$f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S}) = \Bigg[\sum_{v_i \in \mathcal{V}_c} r^{\mathrm{s},v_i}_{\mathrm{node}}(\lambda_{v_i} - P(R_{v_i} > \tau_{v_i})) + r^{\mathrm{p},v_i}_{\mathrm{node}}(X_{v_i})\Bigg] + \sum_{v_i \notin \mathcal{V}_c} r^{\mathrm{p},v_i}_{\mathrm{node}}(X_{v_i}) \tag{11}$$

In the above definitions, $f^{\mathrm{r}}_{\mathrm{path}}(\mathcal{S})$ and $f^{\mathrm{r}}_{\mathrm{node}}(\mathcal{S})$ represent a reward term (r) for *every* path and node based on schedule $\mathcal{S}$, respectively, with generic reward functions $r^{\mathrm{s},\ell_i}_{\mathrm{path}}(\cdot)$, $r^{\mathrm{p},\ell_i}_{\mathrm{path}}(\cdot)$, $r^{\mathrm{s},v_i}_{\mathrm{node}}(\cdot)$, $r^{\mathrm{p},v_i}_{\mathrm{node}}(\cdot)$ that separately reward safety margins (s) and system performance based on timing (p) for paths $\ell_i$ and nodes $v_i$. There are two key points to note in these reward definitions: 1) critical and non-critical paths/nodes contribute separately as non-critical paths/nodes ($\ell_i \notin \mathcal{C}$ and $v_i \notin \mathcal{V}_c$) cannot be rewarded for improving safety margins; and 2) the rewards for performance are given in the most general form

[2] We use RabbitMQ middleware [19] for implementing our computational graph and measuring response/execution times (details in Section IV).

[3] Given the PDF of response times, the term $P(R_{\ell_i} > \tau_{\ell_i})$ can be evaluated using the Cumulative Distribution Function (CDF) [21].

($r^{\mathrm{p},\ell_i}_{\mathrm{path}}(X_{\ell_i})$ and $r^{\mathrm{p},v_i}_{\mathrm{node}}(X_{v_i})$) as the relationship between timing and performance can vary broadly between tasks (we illustrate specific reward functions in Section IV).

Finally, with all terms defined a scheduler can optimize our safety-performance metric defined as a weighted sum of terms (8)–(11), yielding schedules that trade off safety and system performance relative to probabilistic timing constraints.

*Remark 1:* It is important to note for all SP terms defined above, if a hard timing constraint is desired one can simply set $\lambda_{\ell_i} = 0, \lambda_{v_i} = 0$ which enforces sureness of satisfying $R_{\ell_i} \leq \tau_{\ell_i}$ and $R_{v_i} \leq \tau_{v_i}$.

## IV. Evaluation of SP Metric

### A. Construction of the DAG

In order to evaluate our above defined SP metric, we require an SHP-DAG $G$ that represents a *basis* for robotic computing. We argue that all robots must navigate in the environment, perceive the environment, and allocate high-level tasks according to mission objectives. Thus, below we compose an SHP-DAG $G$ using: (1) YOLO-DynaSLAM [22]; (2) RRT-based path planning [23]; (3) model predictive control [24]; (4) vision-based depth estimation [25]; and (5) the traveling salesperson problem (TSP). The implemented SHP-DAG $G$ is depicted in Fig. 2.[4] It consists of one critical path $\ell_1$ and three critical nodes $\{(v_1, CPU), (v_2, GPU), (v_3, CPU)\}$. The data connections among nodes as well as the precedence constraints are implemented using industry-standard RabbitMQ middleware, well-known for its robustness and speed [26]. A brief description and common parameters of the above algorithms are given below. Additionally, as we require input data to execute each algorithm, we specify the test data below.

*1) YOLO-DynaSLAM ($v_1$ in Fig. 2):* YOLO-DynaSLAM is a visual SLAM system with increased robustness to dynamic environments in real-time. In our implementation, we replaced Dynamic ORBSLAM's YOLOv3 block [22] with our own CUDA C-library implementation of YOLOv4 [27] to fully utilize the GPU acceleration of our embedded NVIDIA hardware. We utilize the RGB-D TUM dataset from [22].

*2) RRT Path Planning ($v_2$ in Fig. 2):* Unlike typical RRT-based solutions, we implemented an RRT-based planner that utilizes the GPU to perform the nearest vertex test in RRT [28]. Utilizing the GPU yields $O(1)$ scaling in this test and can improve efficiency significantly when obstacle density is high. For our SP metric evaluation, the number of obstacles is set to 2048 and are generated randomly in a fixed 2D environment.

*3) Model Predictive Controller ($v_3$ in Fig. 2):* Model Predictive Control (MPC) is the problem of controlling a linear time-invariant dynamical system according to some reference state. We utilize constrained linear-quadratic MPC, which solves at each time step the finite-horizon optimal control problem. In our evaluation, the state space parameters are generated based on a UAV model (using OSQP [24]).

*4) Traveling Salesperson Problem ($v_4$ in Fig. 2):* We represent high-level robot objectives with a Traveling Salesperson Problem (TSP), which is assigned as non-critical to safety, but important for performance. TSP can be solved using a number of algorithms each having its advantages and issues. Greedy solutions are fast but can generate sub-optimal solutions which would degrade the overall performance of the robot's mission. Branch and Bound (BnB) methods generate solutions closer to optimal but have very high computational requirements. Thus, increasing the share of execution time TSP receives can allow for significantly improved TSP tours as BnB techniques can be used instead of greedy solutions. The dataset we use for evaluation is the famous Odyssey of Ulysses [29].

*5) Depth Estimation ($v_5$ in Fig. 2):* In our evaluation we utilize fastdepth [25], a fast neural network model for monocular depth estimation. We have extended fastdepth using the darknet framework, with a minor change in activation function from ReLU to Leaky ReLU; we call this model fastdepthv2. We use the NYU dataset [30] for our evaluation.

[4]DAG implementation: https://github.com/caslab-vt/SP-metric-analysis.githttps://github.com/caslab-vt/SP-metric-analysis.git.

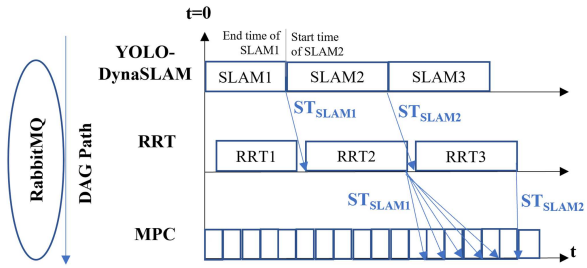


Fig. 3. Multi-rate DAG timing diagram with RabbitMQ message passing for path response time calculations. Each block represents one execution of the respective task.

TABLE II
SP Evaluation Parameters, Where "·" is Function Input

| Parameters | Values |
|---|---|
| $\tau_{v_1}, \lambda_{v_1}$ | 1.36, 0.5 |
| $\tau_{v_2}, \lambda_{v_2}$ | 1.0005, 0.5 |
| $\tau_{v_3}, \lambda_{v_3}$ | 0.025, 0.5 |
| $\tau_{\ell_1}, \lambda_{\ell_1}$ | 5.85 , 0.5 |
| $p_{cp}, p_{cn}$ | $-0.1exp(10 * abs(\cdot))$ |
| $r^s_{node}, r^s_{path}$ | $0.8log(\cdot + 1)$ |
| $r^p_{node}, r^p_{path}$ | $0.2\mathbf{z}(\mathbf{P})$ |

### B. Experimental Setup and Data Acquisition

To evaluate our SP metric on real hardware, the above defined SHP-DAG $G$ is executed on the NVIDIA Jetson AGX Xavier, a widely used platform in robotics. We use the RabbitMQ middleware [26] to implement the DAG edges and calculate the response and execution times. Fig. 3 depicts the timing diagram of our SHP-DAG's critical path and our measurement methodology. Specifically, we utilize RabbitMQ message passing to pass timing information down the DAG path and gather all data at the end of the path for analysis.

Next, we define all parameters in our SP metric to allow for evaluation based on task timing data measured as described above. Table II lists these parameters and choices of penalty and reward functions. The choices for $\tau, \lambda$ have been made based on timing benchmarks of each task from the original authors of each

task algorithm, only modified to match the relatively lower capability of the NVIDIA Jetson AGX Xavier platform. Specifically, it can be reasoned that around 1 s is a general threshold for the SLAM task based on [31]. However, some SLAM algorithms are much faster, such as ORB-SLAM2 at 37 ms [32] and 200 ms for [33]. As we employ YOLO-DynaSLAM [22] that ran at 500 ms per frame on an Intel Core i7-7700 multicore system, we select 1.36 seconds based on the relatively less capable Xavier hardware. Next, the RRT path planning has a wide range of execution times based on the planning environment and task-specific parameters. As our planner is GPU intensive and parallelized, we have seen 0.1 s execution time on more capable hardware [23], [28]. Thus, for the embedded Xavier hardware we set the safety threshold at around 1 s. For the MPC task, 25-100 ms execution times are common [34] and our threshold is set accordingly. Finally the TSP task can have a very wide range of execution times based on the algorithm used [35]. We set the period of TSP at 10 seconds, meaning it executes for the entirety of the DAG activation. Then, depending on the share of computational resources TSP receives over the DAG activation period, the better the TSP tour optimality, which gives us a direct measure of task performance (below).

For the reward/penalty functions, we sought to have diminishing returns of timeliness of tasks for the reward functions, while penalizing exponentially for the violation of safety-based timing thresholds. Finally, performance is measured based on the TSP task's solution optimality, which varies based on share of ET. Specifically, in Table II, $\mathbf{z}(\mathbf{P})$ denotes a normalization function such that $\mathbf{z}(\mathbf{P}) \in [0:1]$ where $\mathbf{P}$ denotes the tour optimality of TSP.

For the evaluation of SP, we set $A_{\text{DAG}}$, the activation period of $G$, to 10 seconds, which is equal to the largest period in the DAG. This produces 10 s windows where we study how the distribution of computational resources across tasks in $G$ influences the value of our SP metric.

### C. Motivating SP: Environmental Correlation of SLAM

We begin our SP evaluation by first corroborating a key motivating factor for the SP metric. Specifically, we argued previously that a robot's computation can vary widely during deployment with influence from the environment [36], motivating the design of our SP metric. In our first experiment, we aimed to demonstrate such variations and show correlation with the robot's environment. Specifically, we executed only the YOLO-DynaSLAM task on the RGB-D TUM dataset [37] and recorded the task response time. At the same time, we recorded the size in pixels of the depth masking region for dynamic elements of the scene used during background inpainting. The depth masking area is calculated using the CPU based on recursion. Thus, as the depth region of moving objects grows larger, the YOLO-DynaSLAM task spends a huge amount of time traversing the recursive loops and hence suffers from high response times. This response time variation and correlation with depth region size is shown in Fig. 4 . It is clearly visible that the peak response times correspond to high pixel sizes of the depth masking region. To verify this claim, we calculated the Pearson Coefficient [38], yielding $r = 0.608$ and p-value $= 4.156e^{-5}$, indicating a correlation between the response time of YOLO-DynaSLAM and the size of dynamics elements in an environment. Most importantly, given the large range of measured response times (peak $> 6$ seconds, minimum $< 1$ second), a WCMS or AMS metric will be insensitive to schedule adjustments, as shown below.

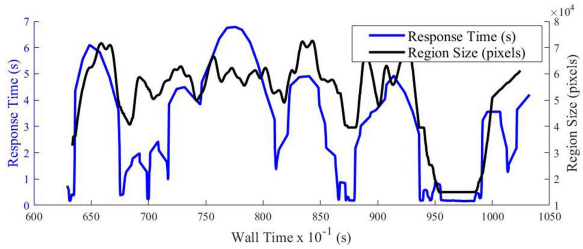


Fig. 4. YOLO-DynaSLAM response time vs. depth masking region size in pixels. The significant correlation between the two parameters indicates the environmental correlation of SLAM response time.

TABLE III
SCHEDULING CLASSES AND POLICIES USED IN EXPERIMENTAL SETUP

| Notation | SLAM | RRT | MPC | TSP |
|---|---|---|---|---|
| S1 | CFS | CFS | CFS | FIFO |
| S2 | FIFO | CFS | CFS | CFS |
| S3 | CFS | FIFO | CFS | CFS |
| S4 | CFS | CFS | FIFO | CFS |
| S5 | CFS | FIFO1 | FIFO2 | CFS |
| S6 | FIFO1 | CFS | FIFO2 | CFS |
| S7 | FIFO1 | FIFO2 | CFS | CFS |
| S8 | FIFO1 | FIFO2 | FIFO3 | CFS |
| S9 | FIFO2 | FIFO3 | FIFO1 | CFS |
| S10 | FIFO3 | FIFO1 | FIFO2 | CFS |
| S11 | FIFO2 | FIFO3 | FIFO4 | FIFO1 |
| S12 | FIFO3 | FIFO4 | FIFO2 | FIFO1 |
| S13 | CFS | CFS | CFS | CFS |

Higher number associated with FIFO indicates higher priority.

### D. Evaluating SP: A Case Study With Linux Schedulers

Motivated by the above observations, we conclude our experimental study with a sensitivity analysis of our SP metric compared to the standard WCMS and AMS metrics. Specifically, as the purpose of a metric is to enable *optimization* of scheduling parameters to improve task timing (c.f., Problem 1), it is critical that a metric is *sensitive* to such parameters to differentiate between schedules and their impact on safety/performance. To begin, recall from Section II-C that two of the primary choices of Linux kernel schedulers for scheduling robot computation are CFS (not real-time) and FIFO (real-time). To demonstrate the effect of these schedulers (and their parameters) on timing in our SHP-DAG $G$, we generated a representative set of schedules with combinations of tasks assigned to CFS and FIFO, see Table III. These schedules represent the possibilities when scheduling our SHP-DAG $G$, ranging from a completely fair schedule (S13), to a completely real-time schedule with task priorities (S11/S12). For each schedule in Table III, we executed our SHP-DAG $G$ on the hardware setup detailed in Section IV-B and collected RT/ET data for all nodes and paths in $G$. We repeated this data acquisition for all schedules to produce a Monte Carlo data set over which we calculated mean

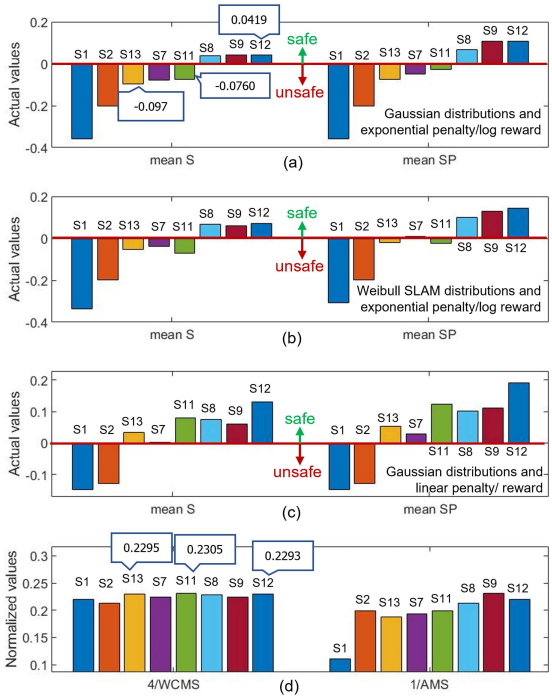


Fig. 5. Comparative SP sensitivity analysis. (a) All distributions are Gaussian and penalty/reward functions exponential and logarithmic as defined in table II. (b) YOLODynaSLAM is Weibull. (c) All distributions are Gaussian but penalty/reward functions are uniformly linear. (d) WCMS and AMS for schedules from Table III. WCMS and AMS are normalized to 4/WCMS and 1/AMS for scale matching. For all bar plots, larger is safer in terms of task timing.

SP, WCMS, and AMS values, the result of which is depicted in Fig. 5.

### *E. Results and Discussion*

In analyzing Fig. 5, we start with some basic observations. First, when comparing the values of SP (which are broken down into safety contribution and overall SP) to WCMS and AMS, it is clear that our SP metric is far more sensitive to schedules as it clearly differentiates between safe and unsafe timing in $G$. For instance, we have highlighted very similar WCMS values for three schedules in Fig. 5 (bottom left), two of which our SP metric deems as unsafe (S11 and S13) and one as safe (S12). Another example of such behavior can be seen in the AMS value for S2 (bottom right), a very unsafe schedule according to SP, yet is very close to safe or nearly safe schedules according to AMS (S8 and S11). Consulting Table III, we see that S2 only allows real-time scheduling of SLAM which helps to cope with the dynamism shown in Section IV-C and improve the AMS value. However, with all other tasks in CFS the overall timing of $G$ suffers with the second lowest SP value.

Next, we can see the effects of changing the probability distribution function from Gaussian to Weibull in Fig. 5(a) and (b). In Fig. 5(a), we have a Gaussian distribution for all the tasks and in (b) we have Weibull distribution for YOLO-DynaSLAM only, which is verified by the response time profile of this SLAM. The result shows a slight change in safety specifications for Schedules S13, S7, S11 and S8. Interestingly, we see S7 goes

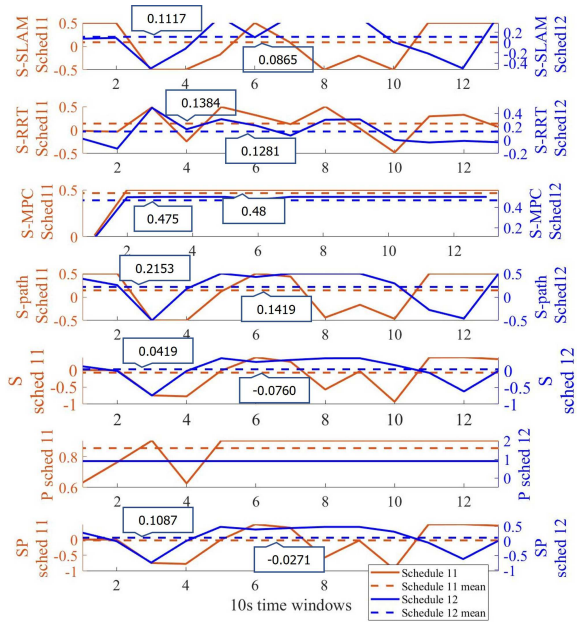


Fig. 6. Distribution of safety (S) values over nodes and paths, as well a overall safety, performance, and SP value. Schedule 11 and Schedule 12 correspond to S11 and S12 in Table III.

from being unsafe in Gaussian distribution to minimally safe in Weibull. Therefore, we can deduce that assuming a Gaussian distribution is the more conservative selection among the two, as it assumes more probability mass in the right-hand tail of the distribution that the timing data for SLAM does not reflect. Lastly, in S7 we see the mean S value is negative while mean SP is positive. This can happen as this is the mean over all the values calculated, recalling that a negative S value *cannot* induce a positive SP in any *instantaneous* measurement by our SP definition.

We further change our penalty and reward functions from being exponential/log to being linear, which again can be seen as conservative versus non-conservative options. We see the effect in Fig. 5(c), where many of the unsafe schedules are deemed safe on average and with much less sensitivity compared to (a) and (b). The explanation here is that, because we have omitted the log (saturating) reward, the linear reward will dominate SP more for task timing when the system is safe ($\mathcal{V}_{\text{us}} = \emptyset, \mathcal{C}_{\text{us}} = \emptyset$). This creates a situation where even though schedulability remains the same (as the underlying task timing in our data has not changed), the mean S and SP of S11 for example can rise substantially as the linear reward dominates in cases where the system is safe. Thus, it is logical to use a saturating reward so that we have the opportunity to balance with performance when the system is safe.

Finally, we conclude our analysis by looking deeper at schedules S11 and S12 in Table III. We select these schedules for further analysis as they assign all tasks to the FIFO scheduler (with varying priorities), which is the expected setup for a real-time application. First, we see from Fig. 5 that the WCMS values for S11 and S12 are effectively the same. Furthermore, comparing to the WCMS value of schedule S13, which is *all CFS* and capable of no real-time guarantees, we see that S11, S12,

and S13 are equivalent. SP on the other hand, clearly identifies S11 and S13 as unsafe, while S12 is safe. To understand this sensitivity of the SP metric, we can study Fig. 6 which shows a single execution of our SHP-DAG $G$ for schedules S11 and S12. Most importantly, as S12 has higher priority for SLAM, we again see that the wide variation in the timing of SLAM is managed better on average, leading to a much improved (and safe) SP value.

*Remark 2:* In Fig. 6, S-RRT shows higher mean safety for schedule 11 compared to schedule 12. This is due to the fact that the priority-safety relationship is not linear in general for complex task scheduling. The non-linear relationship between priority and timing is recognized and expected due to the interconnections and dispatch timing of tasks within the CPU. Indeed, this acts as motivation for a custom scheduler that optimizes SP in selecting task schedules (based on our formulation in Problem 1), which is a key step in future work.

## V. Conclusion and Future Work

The novelty of the SP metric is that it ensures probabilistic safety for all critical computation, and then balances safety and performance. Fig. 5 shows that the SP metric is highly sensitive to FIFO priority, while WCMS and AMS are relatively insensitive. This implies that using our SP metric a scheduler can balance safety and performance with a much higher level of control than traditional metrics. With varying environmental contexts, scheduling can then be adjusted online for a safe system with balanced performance. However, the Linux schedulers are just a testbed for SP evaluation and do not generate optimized SP scores. A custom online scheduler that optimizes the SP metric is the direction of future work.

## References


[1] L. Heintzman, A. Hashimoto, N. Abaid, and R. K. Williams, "Anticipatory planning and dynamic lost person models for human-robot search and rescue," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 8252–8258.

[2] M. Rangwala et al., "DeepPaSTL: Spatio-temporal deep learning methods for predicting long-term pasture terrains using synthetic datasets," *Agronomy*, vol. 11, no. 11, 2021, Art. no. 2245.

[3] D. B. Abeywickrama et al., "On specifying for trustworthiness," 2022, *arXiv:2206.11421*.

[4] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang, "The safety force field," NVIDIA White Paper, 2019.

[5] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," 2017, *arXiv:1708.06374*.

[6] H. Zhao et al., "Safety score: A quantitative approach to guiding safety-aware autonomous vehicle computing system design," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2020, pp. 1479–1485.

[7] T. Victor, K. D. Kusano, T. Gode, R. Chen, and M. Schwall, "Safety performance of the Waymo rider-only automated driving system at one million miles," 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260210721

[8] M. P. Polverini, A. M. Zanchettin, and P. Rocco, "A computationally efficient safety assessment for collaborative robotics applications," *Robot. Comput.- Integr. Manuf.*, vol. 46, pp. 25–37, 2017.

[9] J. Saenz et al., "Methods for considering safety in design of robotics applications featuring human-robot collaboration," *Int. J. Adv. Manuf. Technol.*, vol. 107, pp. 2313–2331, 2020.

[10] F. Vicentini, M. Askarpour, M. G. Rossi, and D. Mandrioli, "Safety assessment of collaborative robotics through automated formal verification," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 42–61, Feb. 2020.

[11] A. Terra, H. Riaz, K. Raizer, A. Hata, and R. Inam, "Safety vs. efficiency: AI-based risk mitigation in collaborative robotics," in *Proc. IEEE 6th Int. Conf. Control Automat. Robot.*, 2020, pp. 151–160.

[12] A. Farrukh and R. West, "smARTflight: An environmentally-aware adaptive real-time flight management system," in *Proc. 32nd Euromicro Conf. Real-Time Syst. Schloss Dagstuhl-Leibniz-Zentrum für Informatik*, 2020, pp. 24:1–24:22.

[13] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "ROSCH: Real-time scheduling framework for ROS," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2018, pp. 52–58.

[14] Y. Tang et al., "Response time analysis and priority assignment of processing chains on ROS2 executors," in *Proc. IEEE Real-Time Syst. Symp.*, 2020, pp. 231–243.

[15] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 191–204, Jan. 2015.

[16] F. Reghenzani, G. Massari, and W. Fornaciari, "Timing predictability in high-performance computing with probabilistic real-time," *IEEE Access*, vol. 8, pp. 208566–208582, 2020.

[17] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate DAG from multi-rate task sets," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2020, pp. 226–238.

[18] S. Wang, R. K. Williams, and H. Zeng, "A general and scalable method for optimizing real-time systems with continuous variables," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2023, pp. 119–132.

[19] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proc. 13th Int. Conf. Embedded Softw.*, 2016, pp. 1–10.

[20] L.-C. Canon and E. Jeannot, "Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 4, pp. 532–546, Apr. 2010.

[21] M. H. DeGroot and M. J. Schervish, *Probability and Statistics*. New York City, NY, USA: Pearson Education, 2012.

[22] J. Bi, Y. Tao, Y. Zhu, L. Chen, and P. Suresh, "Dynamic ORB SLAM," [Online]. Available: https://github.com/bijustin/YOLO-DynaSLAM/blob/master/dynamic-orb-slam.pdf

[23] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3513–3518.

[24] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Math. Program. Comput.*, vol. 12, no. 4, pp. 637–672, 2020.

[25] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, "FastDepth: Fast monocular depth estimation on embedded systems," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 6101–6108.

[26] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proc. 11th ACM Int. Conf. Distrib. Event-Based Syst.*, 2017, pp. 227–238.

[27] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOV4: Optimal speed and accuracy of object detection," 2020.

[28] B. A. Bharmal, "Real-time GPU scheduling with preemption support for autonomous mobile robots," 2021.

[29] M. Grötschel and M. Padberg, "Ulysses 2000: In search of optimal solutions to hard combinatorial problems," 1994.

[30] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 746–760.

[31] J. Zhang and S. Singh, "LOAM: LiDAR odometry and mapping in real-time," *Robot.: Sci. Syst.*, vol. 2, no. 9, pp. 1–9, 2014.

[32] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robot. Automat. Lett.*, vol. 3, no. 4, pp. 4076–4083, Oct. 2018.

[33] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[34] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *Int. J. Adv. Manuf. Technol.*, vol. 117, no. 5/6, pp. 1327–1349, 2021.

[35] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang, "Benchmarking algorithms for dynamic travelling salesman problems," in *Proc. Congr. Evol. Comput.*, 2004, pp. 1286–1292.

[36] A. H. Sifat, B. Bharmal, H. Zeng, J.-B. Huang, C. Jung, and R. K. Williams, "Towards computational awareness in autonomous robots: An empirical study of computational kernels," *Complex Intell. Syst.*, pp. 1–27, 2023.

[37] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 573–580.

[38] I. Cohen et al., "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Berlin, Germany: Springer, 2009, pp. 1–4.