Received: 23 October 2018 | Revised: 11 November 2019 | Accepted: 13 November 2019

DOI: 10.1002/sys.21528

REGULAR PAPER

WILEY

Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints

Anne Collin[1] | Afreen Siddiqi[1] | Yuto Imanishi[2] | Eric Rebentisch[1] | Taisetsu Tanimichi[3] | Olivier L. de Weck[1]

[1]Strategic Engineering Research Group, Massachusetts Institute of Technology, Cambridge, MA, USA

[2]Research and Development Division, Hitachi America, Ltd., Farmington Hills, MI, USA

[3]Advanced Development Center, Hitachi Automotive Systems Americas, Inc., Farmington Hills, MI, USA

Correspondence
Anne Collin, Strategic Engineering Research Group, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA.
Email: anne.collin19@gmail.com

Funding information
Hitachi Automotive Systems Americas, Inc.

Abstract

With the recent progress of techniques in computer vision and processor design, vehicles are able to perform a greater number of functions, and are reaching higher levels of autonomy. As the list of autonomous tasks that the car is supposed to perform grows, two design questions arise: how to group these tasks into modules and which processors and data buses should instantiate these modules and their links in the physical architecture. Both questions are linked, as the processing capacity of the processors influences how centralized the architecture can be, and the modularization influences the overall system latency as well. Furthermore, our interest lies in designing architectures that perform the tasks rapidly, while minimizing cost. This multiobjective problem is intractable without architecture exploration and an analysis tool. This paper presents a linear optimization formulation to capture these tradeoffs, and to systematically find relevant architectures with optimal latency and cost. The results show that enforcing all safety constraints on the architecture leads to a worst case increase of 17% in latency and 18% component cost per vehicle. The increase in latency is significant at the scale of human driver reaction times.

KEYWORDS

autonomy, functional allocation, latency, safety, systems architecture

1 | INTRODUCTION

Advances in autonomous vehicle technologies are rapid. Whether autonomous vehicles (AV) will only result in various technical demonstrators or in viable products for end users will depend on not only the advance of component technologies, but also on their organization into functional systems that possess attractive life cycle properties such as safety, reliability, producibility, and affordability. While much of the current focus of AV technology development is on demonstrating functional capabilities, an important next step is to begin to determine which tradeoffs among the elements of AV system architectures will yield superior overall system performance. Several prototypes were presented during the 2007 DARPA Urban challenge, giving the first insights on which architectures were promising.[1]

For instance, advances in the performance of computer vision algorithms and the ability for some algorithms to successfully perform object recognition at sufficiently high rates is a key enabler in the creation of fully autonomous vehicles. The performance of deep learning algorithms increases with the size of the training data set and the depth of the neural networks, resulting in these algorithms performing more and more operations.[2] The physical architecture of the components supporting these algorithms therefore contains processors with more computing power, in order to perform the same tasks more effectively within the time required for vehicles to be considered autonomous.

This implies some important tradeoffs in system design. Distributing computational tasks on different processors allows for parallelizing the computation, but the transfer of data between processors increases latency[3] (time taken to complete all required computations). If all the computational tasks are carried out on a single processor, processors with high computing capabilities (and cost) will be required. As prototypes of functioning autonomous vehicles turn into viable products going into mass production, quantifying this tradeoff between cost and latency becomes critical. Our work aims to develop an approach for identifying architectures that can perform

at acceptable levels of latency, but are cost-effective for development and manufacturing. In the work presented here, cost is viewed as the addition of individual component costs, which allows a linear formulation of the system cost. This work aims to extend existing methods from embedded systems design to a higher level, at the early phases of design. Unlike most of the existing work in the area, cost considerations are introduced, in addition to the latency calculation, and the performance impact of allowing a largely distributed system, as well as safety constraints, are evaluated. This allows the designer to quantitatively assess the benefits of specific safety measures and weigh them against potential negative impacts on latency, which is another safety relevant metric.

2 | FUNCTIONAL AND PHYSICAL ARCHITECTURES

As future cars are meant to take over an increasing number of tasks from the driver, the embedded systems and the network they form in the car are becoming a complex system. It is no longer the case that each subsystem performed its own task in isolation from the others (e.g., locking the doors after the car started driving). Autonomous cars have to be able to perform a long list of tasks, in various environments,[4] and the output of one task can be useful information for the next. The list of potential hardware to perform these tasks is also long.[5] This paper presents a method to formally design the architecture of embedded self-driving systems for autonomous cars.

The first step is to build a functional architecture of the car, and to understand how tasks interconnect. Several functional architectures have already been proposed.[6,7] Our functional model is not limited to an abstract model, but it assumes execution times and gives performance constraints on the algorithms that will be used to instantiate the tasks listed in the functional architecture.[8] The specific data reflect a realistic functional architecture used in industry for an autonomous vehicle. The functional architecture is defined as a network of tasks, that communicate to one another via messages, whereas processors and data buses are components of the physical architecture. The functional architecture is the equivalent of a graph, where tasks are the nodes, messages are the edges, and the adjacency matrix of the graph would be the problem's design structure matrix (DSM).[9] The design of such a functional architecture can be informed by recent methods in network analysis such as functional dependency network analysis (FDNA),[10] in order to identify the effects of a functional failure in a highly connected architecture such as the one presented here.

For example, Figure 1 shows that the Cognition 2 task cannot receive the latest information until seven other tasks finish their calculations. It is worth noting that a part of the system, as described by this functional architecture, is examined here, and that the control devices are not included in the selection process, for example. In order to create redundancy in the system and increase safety, some tasks are replicated into a degeneracy task; both tasks require the same input and outputs, but they might be carried out by different algorithms or hardware solutions. Identifying the most capability-carrying functions through a network analysis[11] informs the design of a redundancy network of tasks to improve performance and safety in the first stages of the design. For example, the task "Recognition 4," has the degeneracy equivalent "Recognition 4 Degeneracy." The purpose of redundant tasks is to have a function that takes over if the original function fails, whether because of a hardware or a software failure. Each task has an ASIL (automotive safety integrity level), corresponding to how critical it is. A task and its degeneracy equivalent might have different ASILs.

In order to transfer the safety considerations from the functional architecture to the physical ones, two safety constraints are defined: a task and its degeneracy equivalent should not be allocated to the same processor and all tasks should be matched to a processor that has at least the same ASIL.[12] The ASIL metric captures the robustness of a component of the physical architecture in this model. Further robustness analyses, such as the system operational dependency analysis (SODA)[13] can be employed to extend safety considerations, but are not considered in this work.

Latency is defined as the maximum path length in the functional network. In this asynchronous system, it can represent the time between the recognition of an obstacle and a braking action, for example. It is a function of the tasks themselves, the processors they are being computed on, and the messages they need to exchange, and whether these communications are interchip.[3] As an example of the cost–latency tradeoff, one could design a system where all tasks are on the same chip, reducing the latency due to interchip communication but this would require a processor with very high capacities, and which would therefore be very expensive (the architecture would also be nonrobust to hardware failures). Using several processors linked to each other, the designer needs to specify which tasks should be processed in parallel, while respecting the capacity constraints given by data buses, on the messages that the chips need to exchange. The difference with a network flow problem is that, here, the capacity of the nodes and edges of the network are not set initially.

An assumption of this work is that tasks can be freely assigned to a list of provided processors, in order to, in a first instance, remove constraints to reach new, creative designs. A first avenue to explore the design space of physical and functional allocations is to simply enumerate all possible architectures, evaluate the cost and latency of them, and plot their performance to obtain a Pareto front.[14] However, this approach becomes rapidly computationally intractable due to the dimension of the design space in this type of problems.

The functional allocation problem is equivalent to partitioning the set of functions into an unknown number of subsets.[15] The size of the design space for k functions is therefore the Bell number $B(k) = \sum_{m=0}^{k} \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{1} (m-i)^k$, which, for $k = 24$, amounts to approximately 4.46×10^{17}!

As for the physical allocation problem, if the grouping of tasks into m modules communicate through b buses, and if processors, among the p possible, and buses, among the c possible, are to be chosen, $\binom{p}{1}^m \times \binom{c}{1}^b$ have to be enumerated and evaluated. For example, if $m = 9$, $b = 22$, $p = 2$, and $c = 2$, this represents 2^{31} possibilities. As a full-factorial enumeration and evaluation approach of the combination of the two

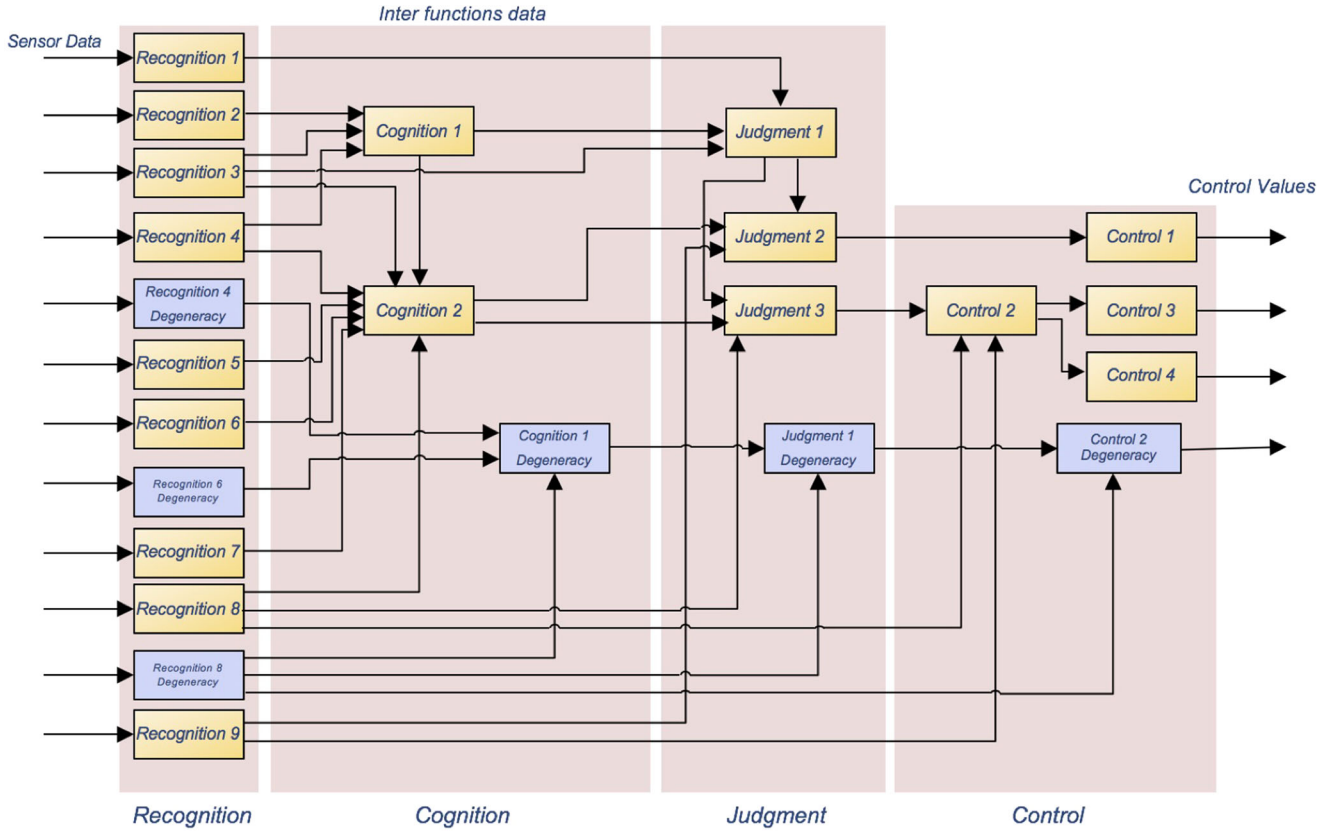

FIGURE 1 Example of a functional architecture for an autonomous car—blue tasks are degeneracy equivalents of some yellow tasks, to improve safety through redundancy

problems is computationally prohibitive, it is necessary to use optimization approaches to systematically and efficiently search for feasible and optimal solutions.

In this paper, we describe a multiobjective discrete optimization model that minimizes cost and latency. We propose that this approach can serve as an initial step in exploring and identifying feasible and optimal architectures for latency and cost metrics—two of the important attributes for autonomous driving systems. The rest of this paper is organized as follows: In Section 3, previous and related work is reviewed; In Section 4, the details of the optimization formulation are presented, and the results obtained through this formulation are shown in Section 5.

3 | RELATED WORK

Architecture principles and models for embedded systems are common in the software and electrical engineering fields, but literature is starting to emerge about how to use these principles at a higher level of the system, and more generally to optimize the architecture of embedded systems specifically for autonomous vehicles.

From an optimization perspective, Ma et al. (1982) give a general algorithm to allocate tasks on a distributed system[16] using binary assignment variables. With a similar point-of-view, other authors proposed ways to design architectures of automotive systems with a computer science perspective.[17–21] Jo et al. (2014) apply the distributed system architecture to this system, and go into the details of the algorithms behind the tasks themselves.[22] Davare et al. (2007) try to minimize latency as a single objective, but they use different design variables.

Practical experiences of building autonomous vehicles, spurred by the DARPA urban challenge, have led to different articulations of requirements to improve safety in autonomous vehicles.[23] It seems that the current approach from the robotics field is to increase safety by improving each of the tasks or sensors individually,[24] for example, including pedestrian intent prediction in path planning,[25,26] whereas experts from the automotive industry have a functional point-of-view on safety.[27,28] Increased safety can also take the form of enhanced communication between the car and the driver, to increase transparency on the reasons why the car makes certain decisions, adding functions to the list of tasks the car has to perform.[29] More elaborate validation methods for safety include probabilistic modeling of the egocar and other agents on the road.[30]

In our work, the optimization problem is formulated with assignment variables, and task attributes, such as activation periods, are assumed to be fixed, in order to reduce the size of the design space and reduce the amount of computation. Zheng et al. (2016) formulate a nonlinear optimization problem to link the software and hardware design, and minimize latency, with reliability, security, and energy as design variables. In a more qualitative analysis, Meng and Zhang

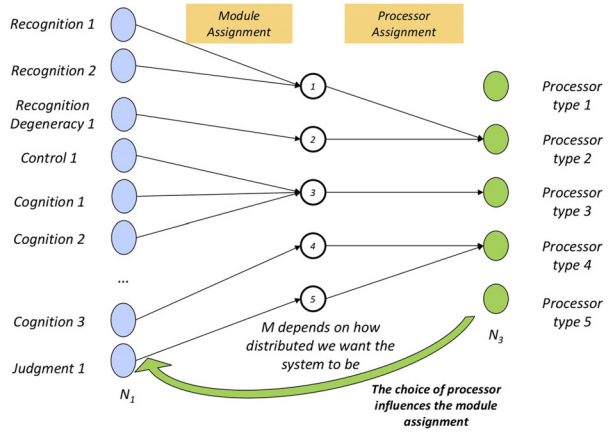

FIGURE 2 Link between functional and physical architecture choices

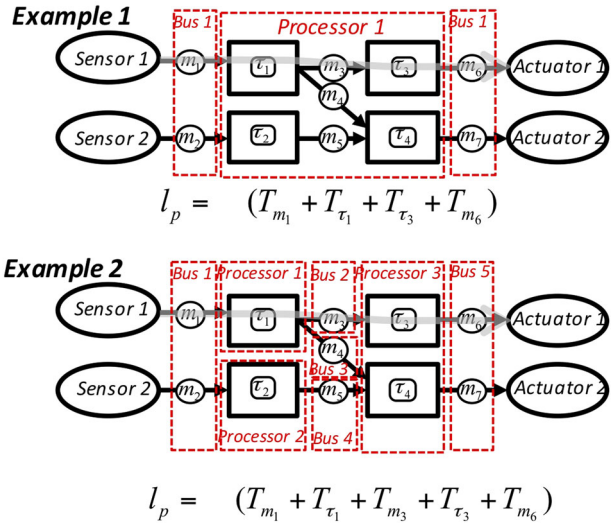

FIGURE 3 Example of latency calculations for two different architectures. The latency is calculated for the path between Sensor 1 and Actuator 1 in both cases. Adding a bus adds latency to the message going from task τ_1 to τ_3. To ease the notation in the figure, messages are here designated with their own index, as opposed to the tasks they are linking, in the equations above; m_3 would be designated as $\tau_1\tau_3$ in the provided latency formula

(2017) also reflect on the sensor architecture and its performance as a function of the environment, especially with regard to safety.

Our work tries to link the performance of the system (proxied by end-to-end latency) and the cost of individual components, to create pareto-optimal, yet realistic, functional, and physical architectures. This is therefore a multiobjective problem, in addition to being multidisciplinary.

4 | LATENCY AND COST OPTIMIZATION

The design problem contains two steps; identifying which tasks need to be grouped together without exceeding the computing power provided by available processors (similar to a multidimensional knapsack problem[31]), where objects have to be chosen to put in which knapsack given several constraints on the knapsack capacities, with the final goal of minimizing latency (similar to a scheduling problem[32]), where jobs are assigned to given resources at a particular time to minimize the overall latency.

Figure 2 illustrates the impact of the physical architecture on the functional grouping, and vice versa. The first part of the optimization can be seen as multidimensional knapsack where tasks need to be bucketed into processors of different capacities. The second part of the problem involves the fact that tasks are linked, and they can precede one another, which resembles scheduling problems. As per Martins' (2013) topology of multidisciplinary optimization formulations, the structure of the problem fits into the All-at-Once problem statement, where all the disciplines are factored in the same optimization problem.[33]

In order to quantify the cost-latency tradeoff, the optimization problem is formulated as a linear, mixed integer programming (MIP), multiobjective optimization problem, with constraints on the computational capacity of the processors. Constraints related to safety requirements (of redundancy) are also included in our formulation. This also allows us to investigate and quantify the impact of safety constraints. The linear formulation allows for using branch-and-bound methods that are implemented through widely used solvers. The design space can hence be explored, and different pareto-optimal architectures can be found for minimizing latency and cost objectives.

As is shown in Figure 2, a module is a theoretical entity that regroups different tasks and that needs to be assigned to a type of physical processor. It is an intermediate step to be able to vary both the functional and the physical allocation at the same time. The grouping is performed by the optimization. M is an external variable that can be changed, and decides how decentralized the architecture is allowed to be. For example, $M = 1$ would lead to all tasks being forced on only one module, whereas $M = 7$ would allow the optimization to distribute the tasks on up to seven different modules, even though fewer might be used.

4.1 | System latency model

The end-to-end latency is the time elapsed between any of the tasks that start at time $t = 0$ and the last task to be finished. The relevant scheduling theory results are detailed in.[34] It can be seen as the longest path between any of the tasks in the left column of Figure 1 and the tasks in the right column. It is composed of the time needed to perform each task on the path, as well as the time to transfer a message from one task to the next one, should they be on two different processors.[35]

The following equation is therefore used for latency (see also Figure 3):

$$\Theta = \max_{\mathcal{P}} \sum_{\tau_i \in \mathcal{P}} (r_{\tau_i} + T_{\tau_i}) + \sum_{\tau_i\tau_j \in \mathcal{P}} (r_{\tau_i\tau_j} + T_{\tau_i\tau_j}), \quad (1)$$

where $\mathcal{P}$ is a possible path in the network, r_{τ_i} (resp. $r_{\tau_i\tau_j}$) the worst case response time for task τ_i (resp. for message $\tau_i\tau_j$), and T_{τ_i} (resp. $T_{\tau_i\tau_j}$) the activation period of task τ_i (resp. of message $\tau_i\tau_j$). Throughout the

model, indices referring to messages are composed of the two tasks that the message is linking. For example, $T_{\tau_i\tau_j}$ is the period of the message linking τ_i and τ_j.

Although this formulation does not capture all the details of the processes taking place in the transmission of messages and the execution of tasks, it linearly links the hardware and logical attributes. It quantifies the tradeoffs presented earlier between centralized and distributed architectures at the functional and physical levels.[22] As the primary goal here is to relatively compare architectures, rather than to determine latency behavior of a single design at high fidelity, we consider this worst case simplifying formulation to be adequate.

A closed-form analytical expression to represent latency in a system composed of tasks and messages that can preempt one another, with potentially buses with different protocols, is not possible.[36] As our purpose is to generate early designs rapidly, the preference is given toward a formulation that leads to a solution of the optimization problem in a reasonable amount of time, to be able to generate a Pareto front. The priorities of the different tasks are neglected in our formulation, as this is usually done at a later stage of the design.[3]

4.2 | Design variables

As encountered in the cited previous work, as well as in the topology of systems architecture problems detailed by Selva,[37] the model uses assignment variables that are binary; the variable is equal to 1 when the two items it represents are associated. Latency is modeled as a continuous dependent variable. The formulation is an MIP problem, that can be solved relatively fast with solvers.

τ_i represents task i, m_k represents the kth module, and π_r represents the rth choice in the list of potential processors.

$\mathcal{M}$ represents the space of existing messages. Messages are designated by the name of the sender task followed by the name of the receiver task. For example, if Task 1 sends a message to Task 3, the message will be denoted as $\tau_1\tau_3$.

- $x_{\tau_i m_k}$ represents the assignment of a task to a module
- $z_{m_k\pi_r}$ assignment of a module to a processor
- $d_{\tau_i\tau_j b}$ represents the assignment of a message to a bus

4.3 | Dependent variables

- $y_{\pi_k\pi_l,\tau_i\tau_j}$ linearization variable, equals to 1 when two communicating tasks are on different processors
- A_{ikr} linearization variable, equals to 1 when task τ_i is assigned to processor π_r through module k
- $U_{\tau_i\tau_j b}$ linearization variable, equals to 1 when message $\tau_i\tau_j$ is assigned to bus b

Parameters for the possible components of the physical and functional architecture are provided in Table 1. The numbers chosen to carry out the optimization are deemed to be representative of the ones available in the industry, and are mostly used with the purpose of illustrating the presented method.

TABLE 1 Parameters for the optimization model

Parameter name	Notation	Units
Task τ_i		
Activation period	T_{τ_i}	seconds
Clock cycle	Cy_{τ_i}	-
RAM usage	RAM_{τ_i}	KB
ROM usage	ROM_{τ_i}	KB
ASIL	$ASIL$(i)	(-)
Message $\tau_i\tau_j$		
Activation period	$T_{\pi_k\pi_l,\tau_i\tau_j}$	seconds
Size	$s_{\tau_i\tau_j}$	bytes
Processor π_k		
Clock rate	Ra_{π_k}	MHz
RAM capacity	RAM_{π_k}	KB
ROM capacity	ROM_{π_k}	KB
ASIL	$ASIL(k)$	(-)
Cost	C_k	-
Bus b		
Transmission Time for a single bit	t_b	seconds per bit
Cost	C_b	-

4.4 | Optimization model and constraints

The cost model C takes into account the cost of the physical components present in a given architecture. The cost expression is made up of two terms; the first one refers to the cost of processors,and the second one to the cost of buses. The index k refers to the choice of physical processor.

$$C = \sum_m \sum_k z_{mk} * C_k + \sum_b \sum_{\tau_i\tau_j\in\mathcal{M}} d_{\tau_i\tau_j b} * C_b. \tag{2}$$

The factor 1000 in (3) comes from time units (seconds), with results ranging from 0.3 s to 0.4 s, and with cost values in the 350–600 cost units range. In order to keep the problem properly scaled, this factor is used in the formulation of the objective function J, which is a usual technique in multiobjective optimization.[38]

$$\min J = 1000 * \Theta + C \tag{3}$$

s.t.

$$\sum_m x_{\tau m} = 1 \quad \forall\tau \tag{4}$$

$$\sum_m z_{m\pi} = 1 \quad \forall m, \tag{5}$$

meaning that each task is assigned to exactly one module, and each module to exactly one processor.

4.4.1 | Processor capacity constraints

For each module, the computation amount, which is the clock cycle of the task (Cy_τ) divided by its activation period (T_τ) (in MHz), the RAM usage and the ROM usage should not exceed the clock rate available, the RAM capacity and the ROM capacity, respectively, of the assigned processor. RAM_k (respectively, ROM_k) designates the RAM (respectively, ROM) usage or capacity of component k in KB.

$$\sum_{\tau} x_{\tau m} * Cy_\tau / T_\tau \leq \sum_{\pi} z_{m\pi} * Ra_{\pi_k} \quad \forall m, \tag{6}$$

$$\sum_{\tau} x_{\tau m} * RAM_\tau \leq \sum_{\pi} z_{m\pi} * RAM_\pi \quad \forall m, \tag{7}$$

$$\sum_{\tau} x_{\tau m} * ROM_\tau \leq \sum_{\pi} z_{m\pi} * ROM_\pi \quad \forall m. \tag{8}$$

4.4.2 | Bus capacity constraints

This constraint ensures that the bus used to transmit the message between task i and task j can fit the number of bytes in the message. $d_{\tau_i\tau_j b}$ represents the assignment of message $\tau_i\tau_j$ to bus b, $s_{\tau_i\tau_j}$ is the size of message $\tau_i\tau_j$, $T_{m_k m_l, \tau_i\tau_j}$ is the activation period of message $\tau_i\tau_j$, and t_b is the transmission time of bus b. The factor 8 comes from the conversion from byte to bit.

$$\sum_{\tau_i,\tau_j} d_{\tau_i\tau_j b} * 8 * \frac{s_{m_k m_l,\tau_i\tau_j}}{T_{m_k m_l,\tau_i\tau_j}} \leq \frac{1}{t_b} \quad \forall \tau_i\tau_j \in \mathcal{M} \quad \forall m_k \neq m_l. \tag{9}$$

4.4.3 | Task precedence constraint

This equation states that, in the worst case, a task j downstream of task i can only start once the following has happened:

- Task i can start.
- Time worth one period of task i has passed, therefore guaranteeing the task has started at least once.
- Task i has been processed on the processor it is being allocated to, and this processing time depends on the task's clock cycle and the processor's clock rate.
- The message going from task i to task j has been transmitted. The formula for this message induced latency comes from bit-stuffing,[39] as is done with CAN buses, using 11-bit identifiers for the messages. The numbers 55 and 10 come from counting the bits in the CAN protocol.
- The period of the message has passed, therefore guaranteeing it has been transmitted at least once.

$$\begin{aligned}\theta(\tau_j) \geq\ & \theta(\tau_i) \\ & + T_{\tau_i} + Cy_{\tau_i} \sum_r \sum_k A_{ikr} * \frac{1}{Ra_r} \\ & + \sum_b U_{\tau_i\tau_j b} * (55 + 10 s_{\tau_i\tau_j}) * t_b \\ & + \sum_{m_k \neq m_l} T_{m_k m_l,\tau_i\tau_j} * y_{m_k m_l,\tau_i\tau_j} \quad \forall \tau_i\tau_j \in \mathcal{M}.\end{aligned} \tag{10}$$

4.4.4 | Latency addition on the path

$$\Theta \geq \theta(\tau_i) + T_{\tau_i} + Cy_{\tau_i} \sum_r \sum_k A_{ikr} * \frac{1}{Ra_r} \quad \forall \tau_i. \tag{11}$$

4.4.5 | Linearization constraints

$\forall m_k \neq m_l$ and $\tau_i\tau_j \in \mathcal{M}$

$$y_{m_k m_l,\tau_i\tau_j} \leq x_{\tau_i m_k} \tag{12}$$

$$y_{m_k m_l,\tau_i\tau_j} \leq x_{\tau_j m_l} \tag{13}$$

$$y_{m_k m_l,\tau_i\tau_j} \geq x_{\tau_i m_k} + x_{\tau_j m_l} - 1. \tag{14}$$

$\forall \tau_i, m_k, \pi_r$

$$A_{ikr} \leq x_{\tau_i m_k} \tag{15}$$

$$A_{ikr} \leq z_{m_k \pi_r} \tag{16}$$

$$A_{ikr} \geq x_{\tau_i m_k} + z_{m_k \pi_r} - 1. \tag{17}$$

$\forall b, \forall m_k \neq m_l$ and $\tau_i\tau_j \in \mathcal{M}$

$$U_{\tau_i\tau_j b} \leq y_{m_k m_l,\tau_i\tau_j} \tag{18}$$

$$U_{\tau_i\tau_j b} \leq d_{\tau_i\tau_j b} \tag{19}$$

$$U_{\tau_i\tau_j b} \geq y_{m_k m_l,\tau_i\tau_j} + d_{\tau_i\tau_j b} - 1. \tag{20}$$

4.4.6 | Safety constraints

Defining safety in autonomous vehicles is a challenging task. For the purpose of retaining a computationally tractable model, the safety notions are reduced here to two constraints on the functional and physical allocations. Through these constraints, the scope of safety is limited to the intrinsic system reliability, rather than examining uncertainty in the environment of the vehicle.[40] This assumes that the system only fails when the physical instantiation of a function (the processor to which it is assigned) fails itself. This reflects the usual literature in fault-tolerant architectures analysis,[41] however, an avenue of future work is to analyze reasons for which the functions would fail to perform correctly, even with functioning processor support. An example of such a failure would be the incapacity to recognize a pedestrian in front of the car, without any hardware failure.

The recommendations from the ISO26262 are leveraged to generate constraints on the optimization problem to represent safety.[28] Therefore, a task and its degeneracy equivalent are forced to be in different modules, and the processor choice has to respect the ASIL of the most critical task in the module. This allows the architecture to be

resilient to physical component failure. For two specific tasks i and j that are meant to be on different processors (and therefore in different modules), this constraint materializes in Equation (21). In Equation (22), the ASIL of processor k needs to be at least as high as the ASIL of task i.

$$x_{\tau_i m_k} + x_{\tau_j m_k} \leq 1 \quad \forall m_k \tag{21}$$

$$\sum_m ASIL(i) * A_{imk} \leq ASIL(k) \quad \forall i \quad \forall k. \tag{22}$$

5 | RESULTS

5.1 | Single architecture example

The functional architecture used in this problem represents the entire set of functions needed to drive a car, all the way from sensing to controls. A Level 4 or 5 of autonomy corresponds to all functions successfully carried out in most or all situations.[27] The architecture contains 24 tasks, linked together by 32 messages. Depending on the context the car is driving in, city center with many pedestrian or highway without many traffic signs to read, some of the functions in the architecture will be more solicited than others. For example, if a task is to read traffic signs correctly, it will be activated more often in cities than on highways. However, the architecture considered here is valid for many different environments, and does not assume a specific scenario—the context only influences how much these functions need to be performed. Even though the latency equations are simplified, and the cost model only takes into account component price, this framework represents the steps that can be taken in the autonomous vehicle industry to plan the future compute power needed.

To generate all the following result, the Julia optimization package JuMP is used as a wrapper around the CPLEX solver, usually well suited for mixed-integer programs. Table 2 shows the list of available processors—and their attributes—used in this optimization. These data have been provided by industry stakeholders and are meant to represent the potential tradeoffs that can occur in processor choice. This table illustrates that the cost is usually proportional to computing capabilities of the processors. Indeed, for the same cost, having a higher ASIL means having a lower capacity and clock rate. A first preprocessing step to reduce computation times is to identify processors that are dominated in all categories, and remove them from the list. For example, processors 10 and 14 have the same capacity attributes, an ASIL of 4, and processor 10 is more than twice as expensive as processor 14, and will therefore never be used. This exercise prompts the designer to make this type of decision, which might not have been made in a systematic manner otherwise.

In one of the nondominated architectures appearing on Figure 4, the degeneracy functions are assigned together to a processor, which is separate from the rest of the physical architecture; this is the most natural solution to respect the constraint expressed in Equation (21).

TABLE 2 Available processors and their clock rate, RAM capacity, ROM capacity, and cost

	Attribute				
Processor ID	*Clock (MHz)*	*RAM (KB)*	*ROM (KB)*	*ASIL (-)*	*Cost (-)*
1	2000	4000000	128000	2	118.18
2	2000	4000000	128000	2	109.09
3	400	2000	8000	4	109.09
4	240	1000	4000	4	109.09
5	180	512	8000	2	27.27
6	1200	1200000	77000	2	109.09
7	480	90000	77000	2	31.63
8	2000	4000000	128000	2	136.36
9	2000	4000000	128000	2	109.09
10	800	4000	16000	4	109.09
11	10	2000	1000	2	2.72
12	288	120000	12000	2	22.73
13	120	448	2000	4	18.18
14	800	4000	16000	4	45.45

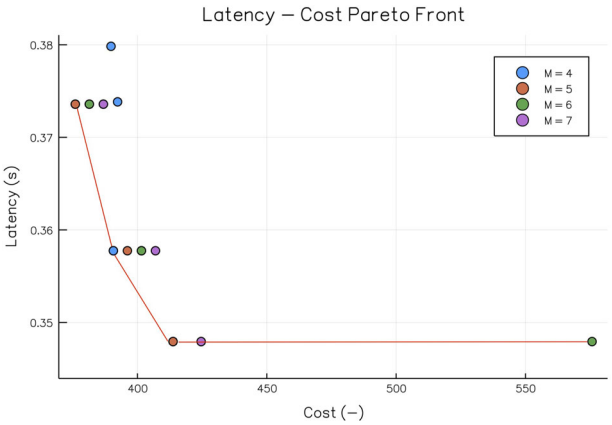

FIGURE 4 Results for different centralization levels—Safety constraints included. The results are calculated within 7% of optimality, meaning that there was a 7% gap between the mixed integer programming solution and the relaxed continuous linear programming solution used for the branch-and-bound. As λ increases, the performance moves to the right

Many recognition functions are allocated individually, as they are heavier and may use a processor on their own.

5.2 | Pareto fronts

The optimization is then run several times by weighing the two objectives cost and latency differently, to see which architectures emerge. This corresponds to optimizing $J = 1000 \cdot \lambda\Theta + (1 - \lambda)C$ for different values of the weight λ. The resulting Pareto Fronts are shown in the following sections. The impact of allowing a more decentralized physical architecture on our two objectives, as well as measure the effects of the safety constraints, are analyzed.

5.2.1 | Impact of task distribution

Figure 4 shows the different architectures obtained by varying M, the maximum number of modules allowed in the solution. The first value of M that leads to a feasible problem is 4: if $M = 1$, the redundancy constraints make the problem infeasible. If $M = 2$ or 3, the ASIL constraints or the processor capacity constraints are causing the problem to be infeasible. Increasing M can only increase the quality of the solution, as the solver can decide to leave empty modules, nonetheless it also increases the computation time; the results shown are within 7% of optimality for $M = 6$ and $M = 7$, hence the seemingly lower performing architectures with a higher M—whereas the optimal value of the objective in this configuration might be the same. This graph shows that increasing M does not improve significantly either the cost or latency of the architecture, meaning that some modules are left empty, and therefore the solution can safely be computed with only four modules, given our current input data, and expect a result close to optimality for all M. The reason why the optimization is run only up until 7% of optimality is that the solver tended to remain in a local minimum for a considerable amount of time before reaching a solution. This is to be expected as problems involving integers are NP-complete, and the solution cannot be guaranteed to be reached in a polynomial amount of time. The usual tradeoff between run time and quality of the solution appeared, and as this tool is meant for early design exploration, the authors felt that cutting the results at this point provided us with enough information for this design stage, in an acceptable amount of time. As the results for $M = 4$ and $M = 5$ are exact, and that the lower bound of the relaxed continuous linear program for $M = 6$ and 7 was very similar to these results, the design space can be explored with lower M values, and not miss significant improvements in the design. The optimization is iterated through several different parameter sets quickly.

Given a value of M, 10 different values of λ yield only three or four different architectures on the Pareto front, meaning that certain architectures satisfy the optimality for different weighing of cost and latency, and lead to superimposed points on the Pareto front. In this case, with our specific set of input data, the computation to optimality only takes about a minute.

The most expensive architecture, costing about 575 cost units, obtained by only optimizing latency, does not decrease the latency significantly compared with the architectures costing 415–425 cost units. This is due to the fact that using more expensive processors with a larger capacity cannot decrease latency indefinitely, as some functions have to wait for others to finish before they can start, and performing some tasks in parallel reduces latency. Forgoing this most expensive architecture, as it wouldn't be a realistic choice if another architecture can satisfy a similar low latency for a much lower cost, the difference between the cheapest and most expensive architecture is of about 100 cost units, or 25% of the cost. This major cost difference comes from the fact that the spread of cost in our list of processors is large, some processors being almost two order of magnitude more expensive than others. From the supplier's perspective, offering a performing processing architecture and reducing cost by a quarter while provides a competitive advantage when advertising to car manufacturers. Currently, 5% of the cost of car models sold with advanced driver assistance systems (ADAS) stems from the added autonomy in the car.[42] This share is likely to increase in the future,[43] meaning that the reduction in cost for this subsystem will have an impact at the system level.

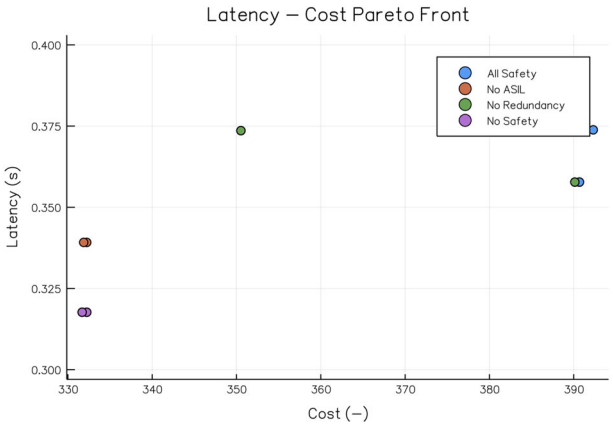

FIGURE 5 Allocations for four modules, with and without safety requirements, zoom on the small latencies

The architecture yielding the lowest end-to-end latency is also more than 10% faster than the architecture with the longest path, and lower cost. Although these numbers are much below average human reaction times for braking, about 1.5 s,[44] a 0.03 s difference in latency represents about 1 m when driving at 100 km/h. Depending on the scenario the car would need to be certified to drive in, this latency reduction might reduce the probability of crashing enough to demonstrate safety. In terms of time metrics for the system, this represents about three times the control period of an assisted driving system, meaning that the latency computed in this work is the main driver of reaction delay, and that a reduction in processing latency is significant and affects the behavior of the autonomous driving system. An avenue for future work would be to test these different architectures in simulations representing different driving scenarios, to quantify the impact of a latency reduction over time, in different settings.

As this work is meant for early design exploration, the gain in latency might not be significant enough to justify the use of a more expensive architecture as a base design, and more effort might be deployed in later design stages to reduce the parameters of each function, such as clock cycle.

5.2.2 | Impact of safety constraints

Figure 5 shows the performance of a four-module architecture with and without each safety constraint. The worst performing architectures with regard to both latency and cost are the ones with all safety constraints enforced (blue points in the upper right corner). Relaxing the ASIL constraints, given the state of our input list of components and their ASIL capabilities, improves cost by more than 15%, and latency by about 7%. However, relaxing the redundancy constraints might reduce the cost to a lesser extent, but not if the ASIL constraint is already relaxed. In terms of safety, reducing the end-to-end latency of

the vehicle is crucial, however the designer has to weigh whether the current decrease in latency justifies a higher risk of failure due to hardware malfunction.

6 | CONCLUSION

This work, linked with broader approaches of architecture and design for system-level properties,[45,46] provides a rapid way of generating physical architectures to support a functional architecture of tasks, taking into consideration both cost and latency. Our results are specific to the characteristics of the functions and hardware used, however the method is applicable to many hardware/software systems, beyond the automotive industry. The intent of this work is to highlight a method to design autonomous driving systems for latency, cost, and safety, rather than present specific results. For parties interested in the specific numerical output of this method, a sensitivity analysis can be conducted to understand the link between the results and the component library that is available to the designer. Safety under the form of hardware constraints increases cost per architecture, but might also increase the latency of the system, therefore decreasing safety in terms of reaction times. An interesting path for future work would be to create a quantitative model of hardware failure risk, and quantify this tradeoff. Furthermore, a more elaborate cost model, beyond the current component costs and taking into account stages of the system from development to retirement, would extend this model to a new application; the analysis of future component development, and which attributes and price points are required to ensure minimal system latency and cost. Considering the lifetime of the system, a dynamic extension of this model could be developed as well, in order to take into account the task attribute changes incurred by software updates, or maintenance events for example.

NOMENCLATURE

Θ	system worst case end-to-end latency (s)
$\mathcal{P}$	path in the architecture, composed of both the functional components (tasks and messages) and the physical components (processors and buses) [-]
r_{τ_i}	worst case response time for task τ_i (s)
$r_{\tau_i\tau_j}$	worst case response time for message $(\tau_i\tau_j)$
T_{τ_i}	activation period of task τ_i
$T_{\tau_i\tau_j}$	activation period of message $\tau_i\tau_j$
l_p	latency of path p (s)
$x_{\tau_i m_k}$	assignment of task τ_i to module m_k (-)
$z_{m_k\pi_r}$	assignment of module m_k to processor π_r (-)
$d_{\tau_i\tau_j b}$	assignment of message $\tau_i\tau_j$ to bus b (-)
$y_{\pi_k\pi_l,\tau_i\tau_j}$	linearization variable, equals to 1 when two communicating tasks τ_i and τ_j are on different processors ($k \neq l$)
A_{ikr}	linearization variable, equals to 1 when task τ_i is assigned to processor π_r through module k
$U_{\tau_i\tau_j b}$	linearization variable, equals to 1 when message $\tau_i\tau_j$ is assigned to bus b
C	overall architecture cost
C_k	cost of component k
Cy_{τ_i}	Clock cycle of task τ_i (-)
RAM_k	RAM usage or capacity of component k (KB)
ROM_k	ROM usage or capacity of component k (KB)
$s_{\tau_i\tau_j}$	size of message $\tau_i\tau_j$ (bytes)
Ra_{π_k}	clock rate (MHz)
Transmission time t_b	(s/bit)
$\theta(\tau_i)$	start time of task τ_i (s)
$ASIL(i)$	automotive safety integrity level (ASIL) of task or processor i (-)
M	number of modules the architecture should contain (-)
N_1	number of tasks in the architecture (-)
N_3	number of different, available processor types (-)

ACKNOWLEDGMENTS

The authors would like to thank Abdelkrim Doufene for his useful comments on system safety, Yukti Matta our project team member, and Virgile Galle from the MIT Operations Research Center for the initial discussions on optimization and knapsack problems.

ORCID

Anne Collin https://orcid.org/0000-0002-7938-9279
Afreen Siddiqi https://orcid.org/0000-0002-3786-5644
Eric Rebentisch https://orcid.org/0000-0003-1124-1312
Olivier L. de Weck https://orcid.org/0000-0001-6677-383X

REFERENCES

1. Buehler M, Iagnemma K, Singh S. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, Vol. 56. Berlin, Heidelberg: Springer; 2009.
2. Canziani A, Paszke A, Culurciello E. An analysis of deep neural network models for practical applications; 2016, http://arxiv.org/abs/1605.07678
3. Davare A, Zhu Q, Di Natale M, Pinello C, Kanajan S, Sangiovanni-Vincentelli A. Period optimization for hard real-time distributed automotive systems. In: *2007 44th ACM/IEEE Design Automation Conference*, San Diego, CA: IEEE; 2007:278-283. http://doi.acm.org/10.1145/1278480.1278553
4. Waymo. On the road to fully self-driving. In: *Waymo Safety Report*; 2017:43. https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report-2017-10.pdf
5. Levison J, Askeland J, Becker J, Dolson J, Al E. Towards fully autonomous driving: Systems and algorithms. In: *IEEE Intelligent Vehicles Symposium IV*; 2011:163-168.
6. Nolte M, Rose M, Stolte T, Maurer M. Model predictive control based trajectory generation for autonomous vehicles. An architectural approach. In: *2017 IEEE Intelligent Vehicles Symposium* (IV),

IEEE; 2017:798-805. http://arxiv.org/abs/1708.02518%0A http://doi.org/10.1109/IVS.2017.7995814http://ieeexplore.ieee.org/document/7995814/
7. Matthaei R, Maurer M. Autonomous driving: A top-down-approach. *Automatisierungstechnik* 2015;63(3):155-167. https://www.degruyter.com/view/j/auto.2015.63.issue-3/auto-2014-1136/auto-2014-1136.xml
8. Schätz B, Voss S, Zverlov S. Automating design-space exploration: Optimal deployment of automotive SW-components in an ISO26262 context. In: *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15 No*. July, New York, NY, USA: ACM Press; 2015:1-6. http://dl.acm.org/citation.cfm?doid=2744769.2747912
9. Browning TR. Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE T Eng Manage*. 2001;48(3):292-306. http://ieeexplore.ieee.org/document/946528/
10. Pinto CA. Functional dependency network analysis. In: *Advanced Risk Analysis in Engineering Enterprise Systems* Boca Raton, FL: CRC Press; 2012:177-256.
11. Guariniello C, Delaurentisa D. Dependency analysis of system-of-systems operational and development networks. *Procedia Comp Sci*. 2013;16:265-274. http://doi.org/10.1016/j.procs.2013.01.028
12. National Instruments, *What is the ISO 26262 functional safety standard?* 2014. http://www.ni.com/white-paper/13647/en/
13. Guariniello C, DeLaurentis D. Supporting design via the system operational dependency analysis methodology. *Res Eng Des*. 2017;28(1):53-69. http://link.springer.com/10.1007/s00163-016-0229-0
14. de Weck OL. *System Architecture Concept Generation. License: Creative Commons BY-NC-SA*. Massachusetts Institute of Technology, MIT OpenCourseWare; 2015. http://ocw.mit.edu
15. Rota GC. The number of partitions of a set. *Am Math Month* 1964;71(5):498. http://www.jstor.org/stable/2312585?origin=crossref
16. Ma PYR, Lee EYS, Tsuchiya M. A task allocation model for distributed computing systems. *IEEE Trans Comp*. 1982;C-31(1):41-47.
17. Coffman EG, Graham RL. Optimal scheduling for two-processor systems. *Acta Informatica* 1972;1(3):200-213.
18. Lo VM. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans Comp*. 1988;37(11):1384-1397.
19. Fernández-Baca D. Allocating modules to processors in a distributed system. *IEEE Trans Software Eng*. 1989;15(11):1427-1436.
20. Zheng W, Zhu Q, Natale MD, Vincentelli AS. Definition of task allocation and priority assignment in hard real-time distributed systems. In: 28th *IEEE International Real-Time Systems Symposium (RTSS 2007)* IEEE; 2007:161-170. http://ieeexplore.ieee.org/document/4408301/
21. Sangiovanni-Vincentelli A, Di Natale M. Embedded system design for automotive applications. *Computer* 2007;40(10):42-51. http://ieeexplore.ieee.org/document/4343688/
22. Jo K, Kim J, Kim D, Jang C, Sunwoo M. Development of autonomous car–Part I: Distributed system architecture and development process. *IEEE Trans Industr Electron*. 2014;61(12):7131-7140.
23. Ozguner U, Stiller C, Redmill K. Systems for safety and autonomous behavior in cars: The DARPA grand challenge experience. *Proc IEEE* 2007;95(2):397-412. http://ieeexplore.ieee.org/document/4142933/
24. Wolcott RW, Eustice RM. Visual localization within LIDAR maps. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* No. Iros; 2014:8.
25. Bai H, Cai S, Ye N, Hsu D, Lee WS. Intention-aware online POMDP planning for autonomous driving in a crowd. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE; 2015:454-460.
26. Lee R, Kochenderfer MJ, Mengshoel OJ, Brat GP, Owen MP. Adaptive stress testing of airborne collision avoidance systems. In: *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)* IEEE; 2015:6C2–1–6C2–13. http://ieeexplore.ieee.org/document/7311450/
27. SAE International, Automated Driving; 2016. https://www.sae.org/news/3544/
28. International Standardization Organization. I*SO26262-2: Road Vehicles—Functional Safety—Part 2: Management of Functional Safety*. International Standardization Organization; 2018.
29. Koo J, Kwac J, Ju W, Steinert M, Leifer L, Nass C. Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance. *Int J Interact Des Manufact*. 2015;9(4):269-275. http://doi.org/10.1007/s12008-014-0227-2
30. Althoff M, Mergel A. Comparison of Markov chain abstraction and Monte Carlo simulation for the safety assessment of autonomous cars. *IEEE Trans Intell Transport Syst*. 2011;12(4):1237-1247. http://ieeexplore.ieee.org/document/5875884/
31. Fréville A. The multidimensional 0–1 knapsack problem: An overview. *Eur J Operation Res*. 2004;155(1):1-21. http://linkinghub.elsevier.com/retrieve/pii/S0377221703002741
32. Graham RL, Lawler EL, Lenstra JK, Kan AHGR. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann Discrete Math*. 1979;5:287-326.
33. Martins JRRA, Lambe AB. Multidisciplinary design optimization: A survey of architectures. *AIAA J*. 2013;51(9):2049-2075. http://arc.aiaa.org/doi/10.2514/1.J051895
34. Stankovic JA, Spuri M, Di Natale M, Buttazzo GC. Implications of classical scheduling results for real-time systems. *Computer* 1995;28(6):16-25.
35. Zheng B, Liang H, Zhu Q, Yu H, Lin CW. Next generation automotive architecture modeling and exploration for autonomous driving. In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Vol. 2016-Septe IEEE; 2016:53-58. http://ieeexplore.ieee.org/document/7560172/
36. Gonzalez Harbour M, Klein MH, Lehoczky JP. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Trans Software Eng*. 1994;20(1):13-28.
37. Selva D, Cameron B, Crawley E. Patterns in system architecture decisions. *Syst Eng*. 2016;19(6):477-497. http://doi.wiley.com/10.1002/sys.21370
38. Papalambros PY, Wilde DJ. *Principles of Optimal Design*. Cambridge: Cambridge University Press; 2000. http://ebooks.cambridge.org/ref/id/CBO9780511626418
39. Davis RI, Burns A, Bril RJ, Lukkien JJ. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Syst*. 2007;35(3):239-272. https://doi.org/10.1007/s11241-007-9012-7
40. de Weck O, Eckert CM, Clarkson JP. A Classification of uncertainty for early product and system design. In: *International Conference on Engineering Design*. Paris, France; 2007.
41. Dominguez-Garcia A, Hanuschak G, Hall S, Crawley E. A comparison of GN&C architectural approaches for robotic and human-rated spacecraft. In: *AIAA Guidance, Navigation and Control Conference and Exhibit* Reston, VA: American Institute of Aeronautics and Astronautics; 2007:15. http://arc.aiaa.org/doi/10.2514/6.2007-6338
42. Meier F. 2019 Subaru Impreza: Slightly Higher Pricing, EyeSight Available on All Trims. Carscom 2018 jul;p. 1. https://www.cars.com/articles/2019-subaru-impreza-slightly-higher-pricing-eyesight-available-on-all-trims-1420700566300/
43. Daziano R, Sarrias M, Leard B. *Are Consumers Willing to Pay to Let Cars Drive for Them? Analyzing Response to Autonomous Vehicles*. Cornell University; 2016.
44. McGehee DV, Mazzae EN, Baldwin GHS. Driver reaction time in crash avoidance research: Validation of a driving simulator study on a test track. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 44 Los Angeles, CA: Sage Publications; 2000:3-320.
45. Siddiqi A, de Weck OL. Modeling methods and conceptual design principles for reconfigurable systems. *J Mech Des*. 2008;130(10):101102.
46. Suh ES, De Weck OL, Chang D. Flexible product platforms: Framework and case study. *Res Eng Des*. 2007;18(2):67-89.

AUTHOR BIOGRAPHIES

ANNE COLLIN is a PhD candidate in the Department of Aeronautics and Astronautics at MIT. Her research aims at building a systems architecture model for autonomous vehicles to provide a quantitative support tool for the choice of sensors and processors in the vehicle. She holds an MS in technology and policy from MIT, as well as an engineering degree from Ecole Nationale des Ponts et Chaussees.

AFREEN SIDDIQI is a research scientist at MIT. Her research is on quantitative modeling of complex sociotechnical systems for planning and design under uncertainty. Her work is focused on space, water, energy, and automotive systems. She has over 75 publications in technical journals including *Systems Engineering*, *Journal of Mechanical Design*, *Journal of Spacecraft and Rockets*, and *Journal of Infrastructure Systems*.

YUTO IMANISHI is a researcher at Hitachi America, Ltd. His research interests include control systems for advanced driver assistance systems, autonomous driving systems, and connected car systems. He is a member of Society of Automotive Engineers of Japan and the Society of Instrument and Control Engineers.

ERIC REBENTISCH is lead researcher at MIT's Consortium for Engineering Program Excellence and lecturer in the MIT Systems Design and Management program. His primary areas of research include closing the strategy-to-implementation gap, high performance enterprise product development, enterprise change management, and system architecting and development strategies.

TAISETSU TANIMICHI is the senior director of Advanced Development Center at Hitachi Automotive Systems Americas, Inc. His development focuses on E/E components of autonomous driving system like ECU, sensors. He has over 20 years of experience in the engineering field of advanced driving assistance systems and autonomous driving systems.

OLIVIER L. DE WECK is a professor of aeronautics and astronautics and engineering systems at MIT. His research focuses on the technological evolution of complex systems over time, both on Earth and in space . He is a Fellow of INCOSE and served as Editor-in-Chief for the *Systems Engineering* journal from 2013 to 2018.

How to cite this article: Collin A, Siddiqi A, Imanishi Y, Rebentisch E, Tanimichi T, de Weck OL. Autonomous driving systems hardware and software architecture exploration: optimizing latency and cost under safety constraints. *Systems Engineering*. 2020;23:327–337. https://doi.org/10.1002/sys.21528