2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00092

AutoE2E: End-to-End Real-time Middleware for Autonomous Driving Control

Yunhao Bai†, Zejiang Wang*, Xiaorui Wang†, and Junmin Wang*
†Department of Electrical and Computer Engineering, The Ohio State University, USA
*Department of Mechanical Engineering, University of Texas at Austin, USA

***Abstract*—The rapid growth of autonomous driving in recent years has posed some new research challenges to the traditional vehicle control system. For example, in order to flexibly change the yawing rate and moving speed of a vehicle based on the detected road conditions, autonomous driving control often needs to dynamically tune its control parameters for better trajectory tracking and vehicle stability. Consequently, the execution time of driving control can increase significantly, resulting in missing the end-to-end (E2E) deadline from detection to computation and actuation, and thus possible accidents.**

In this paper, we propose AutoE2E, a two-tier real-time middleware system that helps the automotive OS meet the E2E deadlines of all the tasks despite execution time variations, while achieving the maximum possible computation precision (and thus minimum tracking errors) for driving control. The inner loop of AutoE2E dynamically controls the CPU utilizations of all the on-board processors to stay below their respective schedulable utilization bounds, by adjusting the invocation rates of the vehicle tasks running on those processors. The outer loop is designed to adapt the computation time and precision of driving control, when the inner loop loses its control capability due to rate saturation caused by vehicle speed changes. Our evaluations, both on a hardware testbed with scaled cars and in larger-scale simulation, show that AutoE2E can effectively reduce the deadline miss ratio by 35.4% on average, compared to well-designed baselines, while having smaller precision loss and tracking errors.

I. Introduction

Recent years have witnessed a rapid growth of autonomous driving, as part of the global wave towards the next generation of artificial intelligence (AI). To date, many car manufacturers have successfully launched products for full or partial autonomous driving, such as Tesla's Autopilot driver-assistance system [1] and Volvo's Pilot Assist [2]. While those autonomous driving products have significantly transformed people's lives, they have also posed some new research challenges to the traditional vehicle control system. One example is their more complex control and computation applications that are designed to handle new tasks like path tracking, stability control, and real-time signal processing. Today's typical vehicle system is already a complicated real-time embedded system with more than 50 Electronic Control Units (ECUs) [3]. For the consideration of cost, autonomous driving applications are often (for now) scheduled to run on the same ECUs with traditional vehicle tasks [4], making the ECU workloads heavier than before. For instance, automated parking or obstacle avoidance are scheduled to run on the same ECU that performs only cruise control before in AUTOSAR, the OS widely adopted for today's automotive ECUs [5].

In traditional vehicle systems, such workload increases can be carefully estimated and scheduled in an offline manner to ensure runtime guarantees of timeliness [6][7]. However, such a static and open-loop scheduling solution may not be suitable for today's self-driving vehicles, because autonomous driving requires much more sophisticated vehicle control whose worst-case execution time (WCET) can be difficult to estimate precisely. For example, the execution time of path tracking can vary significantly for different tracking error requirements. In addition, in order to flexibly change the yawing rate and moving speed of a vehicle based on the road conditions, autonomous driving control often needs to dynamically tune its control parameters for better path tracking and vehicle stability. Consequently, the execution time of autonomous driving control can increase significantly at runtime, resulting in missing the end-to-end (E2E) deadline from detection to computation and actuation, and thus possible accidents. Although it is indeed possible to overestimate the WCETs of autonomous driving applications in a conservative fashion for runtime schedulability guarantees, doing so can cause a significant increase of the number of needed ECUs (and the entire cost) and is usually not desirable to car manufacturers.

In order to address this dilemma, adaptive real-time scheduling [8], [9] is needed to dynamically adjust the workloads of selected vehicle applications, when the execution time of autonomous driving increases at runtime. Such scheduling solutions keep monitoring the vehicle system schedulability online and make necessary workload adaptation in a closed-loop manner to quickly react to execution time variations. To this end, an effective closed-loop scheduling solution is to dynamically control the CPU utilization of the vehicle ECUs [10], [11], [12]. The goal of utilization control is to enforce appropriate utilization bounds (e.g., the Rate Monotonic Scheduling (RMS) bound [13]) on all ECUs, such that all the real-time deadlines of the system can be guaranteed. In a distributed real-time embedded (DRE) system where end-to-end tasks span multiple ECU processors, such as autonomous driving control, a Multi-Input-Multi-Output (MIMO) controller is designed to dynamically adjust the invocation rates of the tasks, such that every subtask can meet its subdeadline so all the tasks can meet their E2E deadlines [11].

Unfortunately, despite their effectiveness in general DRE systems, existing utilization control solutions *cannot* be di-

This work was supported, in part, by NSF under grant CPS-1645657.

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00092

rectly applied to autonomous driving, because they rely solely on adapting the task rates within wide ranges. However, in vehicle systems, the task rate of autonomous driving is mainly determined by the vehicle speed and can only be slightly adjusted above the determined rate. This is because a higher speed usually requires a smaller control cycle (i.e., a higher task rate), in order to adapt to fast changing vehicle dynamics. For example, in order to maintain the same tracking error for the path tracking task when the vehicle speed increases, the task must run more frequently and its invocation rate has to stay above the determined rate [14]. Hence, existing utilization control solutions can cause the task rate of path tracking to quickly reach and saturate at the determined rate, which cannot be further reduced to lower the CPU utilization. As a result, it becomes infeasible for them to control the ECU CPU utilization [15], which can result in undesired deadline misses. Section III tests the discussed scenario for our motivation.

In this paper, we propose AutoE2E, a two-tier real-time middleware system that helps the vehicle OS meet the E2E deadlines despite unexpected runtime execution time variations, while achieving the maximum possible computation precision (and thus minimum tracking errors) for driving control tasks. The inner loop of AutoE2E performs CPU utilization control on all the ECUs, by adjusting the invocation rates of the vehicle tasks running on them. Similar to the existing utilization control solutions, the inner loop can lose its control capability due to rate saturation caused by vehicle speed increases, resulting in undesired deadline misses. When this occurs, the outer loop is designed to dynamically lower the execution time within an allowed range to regain effective CPU utilization control for uninterrupted real-time guarantees. Specifically, this paper makes three major contributions:

- We identify a new research challenge on real-time scheduling of vehicle control systems that is introduced by the recent growth of autonomous driving. After examining traditional open-loop scheduling methodologies used in automotive systems and adaptive real-time scheduling proposed for general DRE systems, we find that a new real-time scheduling solution must be designed due to a special feature of autonomous driving control.
- We design AutoE2E, a two-tier real-time middleware system that overcomes the limitation of existing solutions by having a second-tier controller to dynamically lower the execution time (and so computation precision) within the allowed range to regain effective CPU utilization control for real-time guarantees.
- We evaluate AutoE2E both on a hardware testbed with scaled cars and in larger-scale simulation. Our results demonstrate that AutoE2E outperforms a state-of-the-art solution that relies solely on rate adaptation by 35.4%, on average, in terms of deadline miss ratio, and in the meantime reduces the tracking error of path tracking.

The rest of the paper is organized as follows: Section II discusses the related work. Section III motivates our work by analyzing a typical path tracking scenario. Section IV introduces the design of AutoE2E. Section V presents the evaluation results. Section VI concludes the paper.

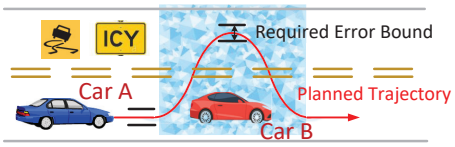


Fig. 1: A possible scenario to cause execution time changes for vehicle tasks in the autonomous driving system.

II. Related Work

Extensive studies have been done to propose various real-time scheduling algorithms for DRE systems with end-to-end tasks [10], [11], [12], [16], [17], [18], [19]. For example, Davareassign et al. have proposed to assign task and message periods for end-to-end tasks using geometric programming in an open-loop manner [16]; Chen et al. use Multi-Parametric Rate Adaptation (MPRA) for discrete rate adaptation in distributed real-time systems with end-to-end tasks given the exact execution time for each task in the system [17]. Greco et al. divide a whole end-to-end task into several segments, and choose how many segments to be executed on each processor [18]. The most closely related work in general DRE systems is EUCON [10], which adjusts the task rate and adopts a model prediction controller to determine the task rate of each end-to-end task based on the current CPU utilization. However, these aforementioned studies all focus on general DRE system whose tasks can be tuned within wide rate ranges. Thus, they assume that there exists at least one feasible solution to guarantee deadlines for tasks in the system. However, in autonomous driving control, a feasible solution may not exist due to the strict execution requirement from high vehicle speed that often results in a tight task rate range.

With its growing scale (in terms of ECUs and tasks) and the wide adoption of the AUTOSAR timing model, today's vehicular driving control system is a typical DRE system with some special timing requirements. Currently, most of the studies on automobile DRE system focus on timing analysis, which is performed offline with fixed execution time of each end-to-end task [20], [21], [22], [7], [23]. For example, Becker et al. estimate the maximum data age under the job-dependency condition [20]. Feiertag et al. introduce the notion of end-to-end timing semantics for the automobile DRE system and provide maximum end-to-end latency with a time path graph approach [23]; Rajeev et al. propose a model-checking technique to compute the worst-case response time and end-to-end latencies [7]. However, these studies focus only on offline scheduling analysis, which cannot handle the unexpected runtime execution time variations introduced by autonomous driving control, despite their effectiveness in traditional automobile DRE system. In sharp contrast, AutoE2E is designed to be an adaptive real-time middleware system that can guarantee the E2E deadlines of all the vehicle tasks despite unexpected runtime execution time variations.

III. Motivation

In this section, we motivate the design of AutoE2E by analyzing a typical scenario of the path tracking application.

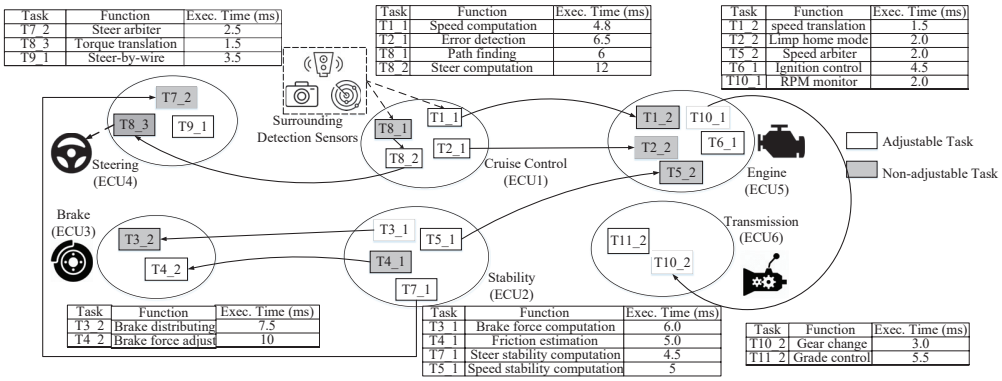
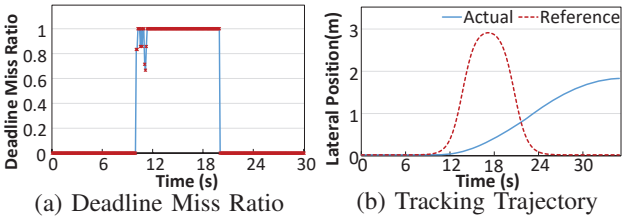


Fig. 2: The workload setup for car A in the motivation experiment, where T8 represents the path tracking application.

As shown in Figure 1, Car A is on the autonomous driving mode, and about to make a double lane change maneuver to pass Car B. The planned driving trajectory and the required tracking error bound are calculated with the path finding application, and shown as the red curve and black line in Figure 1, respectively. The maneuver is done by steering control with a Model Predictive Control (MPC) controller, which calculates the desired steering angle for the vehicle, and sends it to the steering control ECU for actuation. Due to the weather condition, the road ahead is covered with ice so the vehicle stability is harder to maintain with a smaller friction coefficient. In order to let the car track the path and maintain stability, the parameters of the MPC controller must be tuned. Specifically, the prediction horizon is increased to avoid oscillations, and the control horizon remains unchanged to maintain the tracking error. Thus, the total execution time for the path tracking application increases from 12.1ms to 23.5ms for an 18 meter increase in the prediction horizon according to the parameter selection algorithm [24].

To examine this scenario, we also set the most basic vehicle control functions on Car A, such as Anti-lock Braking System (ABS, T4), Traction Control (TC, T5) and Electronic Stability Control (ESC, T6), etc. The descriptions and the precedence constraints of these applications are shown in Figure 2: We choose 11 typical tasks in total and distribute them on six ECUs (numbered ECU1 to ECU6). Each ECU can be considered as an independent processor in the DRE system, and uses the RMS algorithm [13] to schedule the tasks allocated on it. Note that though the typical end-to-end deadline is defined from the data accumulation to the final actuation, the tasks in Figure 2 do not include the sensors and actuators because they are usually firmware whose parameters cannot be tuned online. As a result, the end-to-end deadline for the task shown in Figure 2 is the result of deducting the delays of sensor reading and actuation from the original E2E deadline. The focus application is T8, whose reference path is calculated first with detection sensors (T8_1). Then, the MPC controller is used to calculate the actual steering angle of the vehicle with a variable prediction horizon and control horizon (T8_2). After that, the steering command is sent to the steering control ECU to get the final desired torque for the steering servo (T8_3). We investigate how the MPC execution time variation



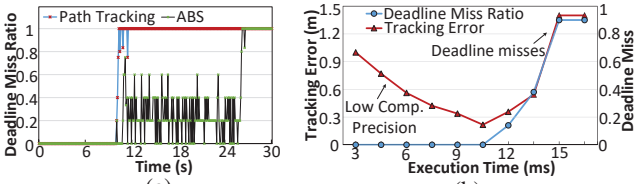
(a) Deadline Miss Ratio (b) Tracking Trajectory

Fig. 3: Given the deadline miss happening continuously, Car A cannot do the pass maneuver given the poor performance of the steering control (T8).

of T8_2 affects the deadline miss ratio of T8, and the tracking performance due to the deadline misses afterwards.

Given the execution time variation for the subtask T8_2 (i.e, steering angle computation), the corresponding deadline miss ratio for the path tracking application T8 is shown in Figure 3(a). We can see that given a large execution time increase of T8_2, the deadline miss ratio increases significantly from zero to nearly 100%. If the computation cannot finish before its deadline, the computation result from T8_2 becomes obsolete and has to be discarded, and the succeeding task cannot update the torque command for the servo to the final mechanical actuator. As a result, the vehicle steering remains unchanged in this control cycle. Given such a high deadline miss ratio, the steering angle cannot be updated in time on the icy road. The corresponding actual trajectory of Car A is shown in Figure 3(b), where we can clearly see the poor tracking result due to these deadline misses. The difference between the reference trajectory and the actual one is so large that Car A might collide with Car B because the steering angle cannot be controlled properly and has to use an obsolete value received several seconds ago due to the deadline misses. Thus, if we rely on the offline scheduling analysis result for the WCET without online monitoring and control, it is possible that the path tracking application performs poorly due to the continuous deadline misses, and these deadline misses can even cause an accident on the road.

However, even with the existing adaptive scheduling solutions, we may still have deadline misses because the task rate for the autonomous driving application is often determined by the vehicle speed. When the vehicle speed increases, the determined task period for the path tracking applications is shortened from 40ms to 20ms, in order to update the control



(a) (b)

Fig. 4: An illustration of saturation. (a): Deadline miss ratio with tight control cycle requirements. (b): The trade off between tracking error and execution time of T8_2. The tracking error increases when the task execution time (i.e., computation precision) decreases (between 3 and 10ms) or when the deadline miss ratio increases (between 10 and 16ms).

commands for the vehicle upon a fixed traveling distance. As shown in Figure 4(a), due to rate saturation, it is infeasible to find a proper task period to guarantee the deadline for our path tracking application under this new determined task period. To solve this problem, a common practice adopted in automotive applications is to shorten the execution time of some tasks such that they can be finished sooner with a lower computation precision and control performance (e.g., a larger tracking error), because a less accurate computation result for a task with reduced execution time is still more desirable than no update at all and consequently a deadline miss [14][25]. However, how to choose execution time (related to the computation precision) for the autonomous application is non-trivial. Figure 4(b) shows the trade-off between the deadline miss ratio and the performance of the steering control application (T8). On one hand, if we do not sufficiently lower the execution time of task T8_2, deadline misses occur and the tracking error can become as large as 1.4m. Such an error is sufficiently large for the vehicle to travel across the lane and causes an accident. On the other hand, if we decrease the execution time too much, the tracking error also increases due to the loss of computation precision. Thus, a novel design is needed to find the optimal value that guarantees the runtime schedulability with the maximum performance for autonomous driving control.

IV. Design of AutoE2E

In this section, we first present the end-to-end task model used in automobile DRE systems. Then, we describe the overview of AutoE2E and introduce each component in detail.

A. System Model

As mentioned in Section I, each control application on an automobile can be modeled as an end-to-end task T_i. Figure 2 shows several examples of periodic end-to-end tasks. For example, task T1 consists of two sub-tasks T1_1 and T1_2. In each task period, T1 first releases one instance of T1_1, and waits for its execution based on the scheduling scheme on ECU1. After T1_1 finishes, an instance of T1_2 is then released on ECU5. Every end-to-end task can adjust its task rate r_i (reciprocal of the task period) in an allowed range $[r_{min}, r_{max}]$. r_{min} is the determined task rate that is set by the vehicle speed. As shown in Figure 5, we assume that there are n ECU processors and m E2E tasks in our target system. Each task T_i has multiple subtasks T_{il} that can run on different ECU processors, where T_{il} is the l^{th} subtask in task T_i. A subsequent subtask cannot be released unless its predecessor is completed. A non-greedy synchronization method, release guard [26], is used to enforce the precedence constraints between subsequent subtasks. As a result, all the subtasks of an E2E task T_i have the same task rate determined by the first subtask. Each subtask T_{il} has an estimated maximum execution time c_{il} measured offline that can vary at runtime. An adjustable subtask T_{il} can adjust its computation precision by shortening its execution time to make the task set feasible for task rate control. We denote a_{il} for the execution time ratio for subtasks T_{il}, which is defined as the quotient of the actual execution time divided by the maximum estimated execution time c_{il}. a_{il} also has a tunable range $[a_{min,il}, 1]$, where $a_{min,il}$ is set based on the specific autonomous driving scenario. For example in Figure 2, $a_{min,il}$ for the stability control T7_1 should be higher when the vehicle detects a sudden decrease of the friction coefficient than that on a normal condition road. For the non-adjustable subtasks whose execution time is constant, their $a_{min,il}$ is set to be 1. We assume that the execution time ratio for different subtasks in an end-to-end task can be tuned individually because each subtask may react to the environment change differently.

To guarantee the deadline and maximize the precision for the control applications, A well-known approach is to enforce the ECU processor utilization bound, such as the RMS bound [13]. The subdeadlines of all the subtasks on a ECU processor are guaranteed if the utilization of the ECU processor stays below its utilization bound. If all the subdeadlines are met, then the end-to-end deadline can be met[1]. Thus, the goal for our system is to control the utilizations of the ECU processors under their respective utilization bounds by adjusting the task rates r_i and execution time ratio a_{il}. Specifically, the problem can be expressed as follows: given a fixed ECU processor utilization bound $\mathbf{B} = [B_1, B_1, ..., B_n]$, the allowable task rate ranges $[r_{min,i}, r_{max,i}]$, and the execution time ratio ranges $[a_{min,il}, 1]$, to dynamically choose task rates $r_i(k)$ and the execution time ratios $a_{il}(k)$ such that in the k^{th} control period, the differences between B_j and the ECU processor utilization $u_j(k)$ for all the processors are minimized:

$$\begin{aligned} \min_{r_i(k)} & \sum_{j=1}^{n} (B_j - u_j(k))^2 \\ s.t. \quad & r_{min,i} \le r_i(k) \le r_{max,i} \\ & a_{min,il} \le a_{il}(k) \le 1 \\ & u_j(k) \le B_j \end{aligned} \tag{1}$$

where the rate constraints ensure that all the task rates stay above their determined task rates, and the same requirements apply to the computation precision constraints for every subtask; the utilization bound constraints ensure that all the end-to-end tasks can meet their deadlines. The utilization $u_j(k)$ of ECU processor P_j can be estimated with $r_i(k)$ and $a_{il}(k)$ as:

[1]We assume network delay is negligible in this model. When network delay must be considered, we can deduct it from the E2E deadline of the task.

$$u_j(k) = \sum_{T_{il} \in S_j} c_{il} a_{il}(k) r_i(k) \tag{2}$$

where S_j is the set that includes all the subtasks allocated to ECU processor P_j. We then define the estimated utilization change $\Delta b_j(k)$ for ECU processor P_j as:

$$\Delta b_j(k) = \sum_{T_{il} \in S_j} c_{il} a_{il}(k) r_i(k) - \sum_{T_{il} \in S_j} c_{il} a_{il}(k-1) r_i(k-1) \tag{3}$$

Since $\Delta b_j(k)$ is derived with the estimated execution time c_{il}, which can be inaccurate at runtime. We use g_j to represent the uncertainty of the execution time of the subtasks on ECU processor P_j. In each control period, $u_j(k)$ can be updated as:

$$u_j(k+1) = u_j(k) + g_j \Delta b_j(k) \tag{4}$$

For example, $g_j = 2$ means that the actual change to the utilization of P_j is twice the estimated change. Note that g_j is *unknown* due to the unpredictability of the execution time for these autonomous driving applications at runtime. Though we cannot know g_j, the stability analysis still ensures that our controller can let the $u_j(k)$ converge to the utilization bound, as long as g_j stays within a certain range (see Section IV.C).

B. Overview

Since Equation (3) involves the multiplication of $a_{il}(k)$ and $r_i(k)$, the system is nonlinear and thus hard to analyze. In order to linearize and simplify the system, we use a two-tier controller design, where two separate controllers are used and aim at controlling one of the two variables (i.e., $a_{il}(k)$ and $r_i(k)$) independently, and treat the other variable as a constant. Thus, Equation (3) can be viewed as a linear model in the perspective of each controller. The impact of the inner-loop controller on the outer-loop controller can be modeled as variations in its system model, and vice versa. Based on the analysis in [27], the stability can be guaranteed for the two-tier controller as long as both controllers are stable.

Based on the linearized system model, we present the design of AutoE2E. As shown in Figure 5, AutoE2E consists of two major components: an inner rate-based control loop (the yellow box on the top in Figure 5) and an outer precision-based control loop (yellow boxes in each ECU processor in Figure 5). In each inter-loop control period, the inner-loop controller is invoked to adjust the task rate for every task in the automobile distributed system based on the current ECU processor utilization collected by the utilization monitor. With a period that equals several inter-loop control periods, the outer-loop controller is invoked to manipulate the execution time ratio of each adjustable subtask on each ECU processor while optimizing the overall computation precision when the task rate requirements become stringent, and restores the computation precision when the requirements become relaxed.

Inner Rate-based Control Loop: The inner rate-based controller consists of a centralized controller for the whole system, with utilization monitors and rate modulators on individual ECU processors. Its control period is set to include several instances of E2E tasks to ensure that the utilization monitor can sample the ECU processor utilization correctly. In each control period, the centralized controller first collects current

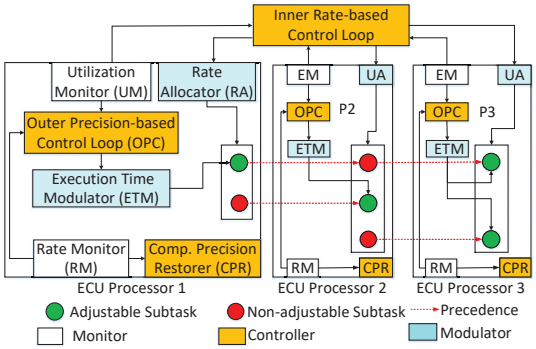


Fig. 5: The overview of AutoE2E. The main components include two control loops: The outer precision-based control loop and inner rate-based control loop.

ECU processor utilizations with utilization monitors. Based on the collected utilizations and the utilization bounds, the controller calculates the task rate for each control application in the next control period, and sends it to the rate modulator on each ECU processor. When the rate modulator receives the updated task rate, it changes the rate of first subtask of each E2E task. Due to the adoption of release guard [26], the rates of the subsequent subtasks are changed consequently.

Outer Precision-based Control Loop: The precision-based controller serves as the outer-loop controller and has a control period sufficiently long to ensure that the it can estimate the utilizations of the ECU processors correctly based on the settled task rates. Each ECU processor has one precision-based controller that manipulates the execution time ratios a_{il} for these subtasks on it. When it is infeasible to control utilizations by the inner-loop controller because the utilization monitor shows that the settled ECU processor utilization is higher than the utilization bound by a preset configurable threshold for several consecutive inter-loop control periods, this controller is activated and works as follows: 1) calculates the difference between the current utilization and the utilization bound for this ECU processor, 2) computes a_{il} for each subtask allocated on this ECU processor based on a reversed relaxed knapsack problem, and 3) changes the execution time ratios for the subtasks with the execution time modulator. Reversely, When the task rate requirement gets relaxed, the computation precision restorer shall decrease the task rate to make the system under-utilized. Then the precision-based controller is activated and increases the computation precision for subtasks on each ECU processor until the utilization monitor detects saturation or the computation precision is fully restored.

Our outer precision-based controller has two key features: First, the controller can prevent saturation of the inner-loop controller due to the high determined task rates given a high speed scenario, and guarantee end-to-end deadlines by decreasing the execution time ratio at a minimum loss of computation precision; Second, it can also restore the computation precision when the determined task rates decrease, even if the current ECU processor utilization is not changed. In the following, we first introduce the outer precision-based controller because it is a major contribution of our paper.

C. Outer Precision-based Control Loop

The outer precision-based controller helps the inner-loop controller achieve the desired utilizations on a coarser timescale by manipulating the execution time ratio a_{il}, while maintaining the computation precision as high as possible. Specifically, the outer precision-based controller has two major goals: 1) decreasing the execution time for the control applications to prevent rate saturation; 2) restoring computation precision if the task rate requirements become relaxed. In this section, we discuss how this controller achieves these two goals, and analyze its stability with standard control theory.

1) Rate Saturation Prevention: For the inner rate-based controller, rate saturation is defined as the scenario where the ECU processor utilizations cannot be controlled to the utilization bounds, given that some task rates are already at their minimum value. For a general DRE system, the rates of end-to-end tasks can be tuned within wide ranges, so the feasibility for processor utilization control is usually assumed to be true [10][17]. However, for the automobile DRE systems, the control cycles of autonomous driving applications are usually determined by the vehicle speed (e.g., the adaptive cruise control must use a high task rate in a fast moving vehicle for speed update, in case of a sudden braking). Given a high vehicle speed, the utilizations of the ECUs can violate their schedulability bounds because the task rates of autonomous driving applications do not have wide ranges and can quickly saturate at their minimum value (i.e., the determined rates). In the automobile industry, the widely used solution for saturation prevention is decreasing the computation precision of some control applications by reducing their execution time [14][25]. Generally, a greater execution time decrease yields a higher possibility to be feasible for the inner rate-based controller, but at the cost of lower computation precision, and vice versa. Thus, for the precision-based controller, we try to optimize the overall computation precision for all the subtasks on each ECU processor while keeping the tasks feasible for the inner rate-based controller given the new determined rates $r_{min,i}$. Mathematically, the problem can be formulated as:

$$\begin{aligned} \max_{a_{il}} & \sum_{j=1}^{n} \sum_{T_{il} \in S_j} w_{il} a_{il} \\ s.t. \quad & r_{min,i} \leq r_i \leq r_{max,i} \\ & a_{min,il} \leq a_{il} \leq 1 \\ & \mathbf{Fr} \leq B \end{aligned} \tag{5}$$

where matrix $\mathbf{F}$ is the execution time matrix and $\mathbf{F}_{il} = c_{il} a_{il}$; w_{il} is the weighting term for subtask T_{il} and is determined by priorities of the autonomous driving applications in the system, e.g., the speed controller has greater weight than the steering controller in the adaptive cruise control scenario. $w_{il} a_{il}$ is the computation precision for subtask T_{il}. Here we assume the computation precision is linearly related to the execution time ratio a_{il} based on the observation in [19]. However, this optimization problem cannot be directly solved because the real execution time for subtask T_{ij} is unknown at runtime. If solving it using the offline estimated execution time c_{il}, we could end up with a wrong result at runtime due to unknown fluctuations as shown in Section III.

Instead of solving the optimization problem directly, the outer precision-based controller uses a feedback mechanism to get a_{il} in the $(k+1)^{\text{th}}$ control period. The reason for using the feedback controller is its stability: Though the real execution time of T_{il} is different from the estimated value c_{il}, the feedback mechanism can still guarantee the convergence to the utilization bound as long as the variation of c_{il} is within a certain range. Since a_{il} only impacts the execution time for a subtask on one ECU processor, changing a_{il} in one specific ECU processor does not affect the utilizations of other ECU processors in the same DRE system. Thus, the utilization of each ECU processor can be modeled independently from others in terms of a_{il}. If we define the decrement of the execution time ratio as $\Delta a_{il}(k) = a_{il}(k+1) - a_{il}(k)$ for subtask T_{il}, according to Equations (3), (4), and the linearized system assumption (i.e., task rate $r_i(k)$ is a constant r_i for the outer-loop precision-based controller), the utilization for P_j in the next control period $u_j(k+1)$ with $\Delta a_{il}(k)$ is:

$$u_j(k+1) = u_j(k) + g_j \sum_{T_{il} \in S_j} \Delta a_{il}(k) c_{il} r_i \tag{6}$$

To minimize the difference to the utilization bound B_j and derive $\Delta a_{il}(k)$ from Equation (6), $u_j(k+1)$ is chosen as the utilization bound B_j. The utilization difference $e_j(k)$ between B_j and $u_j(k)$ for P_j can be expressed as:

$$e_j(k) = g_j \sum_{T_{il} \in S_j} \Delta a_{il}(k) c_{il} r_i \tag{7}$$

where g_j is the uncertainty for the execution time at runtime. Unfortunately, we still cannot use this model to design the controller because g_j is unknown at design time. However, if we assume $g_j = 1$, we can still design the controller based on the approximate execution time c_{il}. As a result, for the real system where $g_j \neq 1$, the controller can behave differently. For this situation, we can show that the controller can still remain stable as long as g_j is within a certain range, and this range can be derived using stability analysis of the closed-loop system by considering the model variations. In each control period, in order to maximize the computation precision for the subtasks on ECU processor P_j (i.e., to minimize the decrement for the computation precision), we need to solve the following optimization problem for $\Delta a_{il}(k)$:

$$\begin{aligned} \min_{\Delta a_{il}(k)} \quad & \sum_{T_{il} \in S_j} w_{il} \Delta a_{il}(k) \\ s.t. \quad & e_j(k) = \sum_{T_{il} \in S_j} \Delta a_{il}(k) c_{il} r_i \\ & 0 \leq \Delta a_{il}(k) \leq a_{il}(k) - a_{min,il} \end{aligned} \tag{8}$$

This is a reversed relaxed knapsack problem [28], where our goal is to minimize the total profit with $\Delta a_{il}(k)$. The terms w_{il} and $c_{il} r_i$ can be seen as the profit and cost for each item ($\Delta a_{il}(k)$), respectively. To solve this problem, we first sort $\Delta a_{il}(k)$ by the profit-cost ratio $w_{il}/(c_{il} r_i)$ in the ascending order, and try to decrease $\Delta a_{il}(k)$ from the first item to the last until the container (i.e., $e_j(k)$) is fully filled. After solving the problem with $\Delta a_{il}(k)$, we can get the new execution time

ratio $a_{il}(k+1) = a_{il}(k) - \Delta a_{il}(k)$. With our assumption $g_j = 1$, the setting point for inner rate-based controller shall be on the determined task rate $r_{min,i}$ for some tasks, and this is still on the edge of saturation due to the lack of margins. Thus, in the real implementation, we leave some margin for variance tolerance of $a_{il}(k+1)$ by slightly increasing $e_j(k)$, to let the inner rate-based controller converge to the task rates slightly higher than r_{min}. Thus, we can ensure that AutoE2E does not suffer from saturation with the new $a_{il}(k+1)$.

2) Stability Analysis: Here we analyze the stability of our precision-based control loop when the estimated execution time is different from the real execution time at runtime, i.e., $g_j \neq 1$. According to Equation (3) and (4), given the current ECU processor utilization $u_j(k)$, the utilization bound B_j, $u_j(k+1)$ can be expressed as:

$$u_j(k+1) = u_j(k) + g_j(B_j - u_j(k)) \tag{9}$$

With Z-transform, the transfer function of our closed-loop system at runtime can be depicted as:

$$G_z(k+1) = \frac{g_j}{z - (1 - g_j)} \tag{10}$$

Based on control theory, the controller is stable if all the poles are within the unit circle. Hence, the system remains stable if $0 \leq g_j \leq 2$. This analysis shows that the our outer precision-based control loop can effectively handle the execution time variations and minimize the utilization error, as long as the summation of actual execution time for all subtasks on one ECU processor is less than twice the summation of their estimation value c_{il}. The detailed analysis and its implication can be found in our technical report [29].

3) Computation Precision Restorer: If the current ECU processor utilizations are at the utilization bounds, both the inner-loop and the outer-loop controllers shall not change their manipulated variables. However, the current stable state may be achieved by reducing execution time ratios of some control applications. When the determined task rates become lower, the execution time of these subtasks should be restored to their original values to allow a better control performance. Unfortunately, the computation precision of the system can only be reduced for saturation prevention, but never restored for the reversed scenarios because the decrease of r_{min} does not cause under-utilization of the system. Thus, when the requirements of control performance for autonomous driving applications become stringent with larger $a_{min,il}$ (e.g., a small required tracking error) in the future, the whole system may become infeasible to find a_{il} in the range $[a_{min,il}, 1]$.

The computation precision restorer is designed to resume the computation precision while keeping the inner rate-based controller unsaturated when the vehicle decelerates. Generally, when the rate monitor detects that the determined task rates have been lowered due to deceleration, the restorer is activated and starts to decrease the task rates while keeping the execution time ratios unchanged. As a result, some processors become under-utilized, and the outer precision-based controller shall be activated to handle the under-utilization scenario: For Equation (8), $e_j(k)$ becomes negative for an under-utilized ECU processor, and the outer precision-based controller shall yield a negative $\Delta a_{il}(k)$, i.e., to increase the execution time ratio for some subtasks. Afterward, the inner-loop controller adjusts the task rate with the new execution time $a_{il}c_{il}$ for each subtask T_{il}, and tries to control the utilizations of ECU processors to their utilization bounds. In the next outer-loop control period, the restorer shall decide whether to further lower task rates based on the saturation status: If the system is not saturated, the restorer shall lower the task rates to make the system under-utilized again. Otherwise, the restorer shall stop decreasing the task rates and wait for the precision-based controller to lower a_{il} due to saturation.

The pseudo code for the computation precision restorer is shown in Algorithm 1. First, when the rate monitor detects that the task rate requirements become relaxed, the computation precision restorer shall decrease all the task rates (Line 1) to the middle point from the current task rates r' and r_{min} for the end-to-end tasks. The reason to use the middle point is that if the task rate r is farther from r_{min}, the inner rate-based controller has a higher probability to converge to a feasible solution [15]. Compared with using a fixed stepsize to decrease r, the stepsize of our algorithm becomes smaller when the task rates r approaching r_{min}, which yields a smaller gap to the optimal computation precision for the subtasks and also a smaller number of iterations to get the final a_{il}. Once the task rates are changed, the system becomes under-utilized and a_{il} shall be increased by the outer precision-based control loop (Line 2). After that, the inner-loop controller tries to control the ECU processor utilizations by adjusting task rates under the new execution time for each subtask. In the next outer-loop control period, the restorer checks whether the system is saturated: If so (i.e., the rates for some tasks are at $r_{min,i}$), this indicates that current a_{il} is too large. Thus, the restorer stops decreasing the task rates, and waits for the outer-loop controller to resolve the saturation by decreasing a_{il} (Line 6). After the ECU processor utilizations have converged to the utilization bound B_j, the restoration process finishes. Otherwise, the restorer further checks whether the execution time ratios have been fully restored: If so, the algorithm finishes as all the subtasks can run at their full computation precision. If not, the restorer shall go back to Line 1 and continue to decrease the task rates (Line 11) until saturation happens or all $a_{il} = 1$.

With the above scheme, the computation precision can be restored in a short time: In our experiment, two rounds of rate decrease are usually sufficient to restore a_{il}. Though there is still gap between the optimal value and ours, this gap is small according to our experiment (see Section V.B). In the real implementation, we can allow some leeway for the restorer to ensure that it is activated only when there is a large drop of the determined rates. Thus, our restorer can avoid chasing the variable r_{min} and work correctly to restore the computation precision with a fixed r_{min} during the restoration process.

D. Inner Rate-based Control Loop

In this paper, we adopt EUCON [10] as the inner rate-based control loop. Based on its Model Predictive Controller

Algorithm 1 The Computation Precision Restorer

Input: Current task rate r', minimum task rate r_{min}
Output: Subtask execution time proportion a_{il}
1: Update task rate r as $r = \frac{r'+r_{min}}{2}$
2: Activate outer precision-based control loop to increase a_{il}
3: Activate inner rate-based control loop with new a_{il}
4: In the next control period of the outer-loop controller:
5: **if** task rate saturated at r_{min} **then**
6: wait for the outer-loop controller to decrease a_{il}
7: **exit**
8: **else if** all $a_{il} = 1$ **then**
9: **exit**
10: **else**
11: **goto** Line 1
12: **end if**

(MPC), EUCON optimizes a cost function representing the tracking error between the current ECU processor utilization and utilization bound. The cost function represents the tracking error within the prediction horizon P and the control penalty within the control horizon M. The cost function $V(k)$ is:

$$V(k) = \sum_{i=1}^{P} \|\mathbf{u}(k+i|k) - \mathbf{ref}(k+i|k)\|^2 + \sum_{i=1}^{M} \|\mathbf{\Delta r}(k+i|k) - \mathbf{\Delta r}(k+i-1|k)\|^2 \quad (11)$$

where $\mathbf{ref}(k)$ is the reference trajectory for the ECU processor utilization, which converges to the utilization bound with an exponential curve; $\mathbf{\Delta r}$ is the change in the task invocation rate. With the feedback mechanism, the close loop system shall converge to the utilization bound if the system is stable. The detailed design and analysis of EUCON are available in [10].

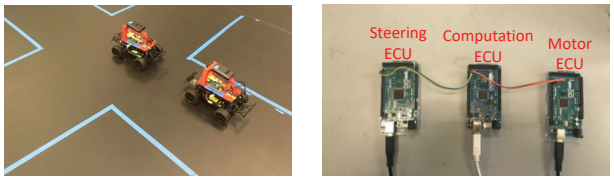
E. Discussions

1) Network Delay: The ECUs in the vehicle are usually connected with different type of buses like as CAN and MOST. Those buses can incur a delay comparable with execution time and task period for the control applications. We can deduct the expected network delay from the original upper bound of the task period to ensure meeting the E2E deadlines. Modeling the network delay is out of the scope of this paper, and we direct interested readers to several studies such as [16][30].

2) Discrete Execution Time Ratio: AutoE2E assumes that the execution time ratios a_{il} can be adjusted continuously. However, some control applications can only allow discrete executive time ratio options. To handle the discrete execution time ratio, we first assume that the variables are continuous, and use a floor operation to get the discrete execution time ratios for those control applications. Though there are known methods to handle discrete parameters in DRE systems [17], they often require priori knowledge of accurate execution time for the tasks at design time, which can have large variations at runtime. Moreover, the complexity to solve a Mixed Integer Linear Programming (MILP) problem is often exponential, which is not acceptable for a real-time system.

V. Evaluation

In the evaluation section, we first introduce the experiment setup. We then test AutoE2E on a hardware testbed with scaled cars (Sections V.B to V.C). Then we test AutoE2E in larger-scale simulation with a realistic vehicle task set (Section V.D). We also measure the overhead of AutoE2E on our testbed, which is less than 10ms. The complete experimental results can be found in our technical report [29].



(a) The Scaled Cars (b) ECU Processors in Use
Fig. 6: Hardware testbed of AutoE2E using scaled cars.

A. Hardware Testbed Setup

Our testbed consists of two 1:16 scaled cars as shown in Figure 6(a). Each scaled car is equipped with a Pulse Width Modulation (PWM) based steering system and an electric motor. We set the speed for the two scaled cars as 70cm/s (25mph for the real vehicle). The ECU processors used for each scaled car are shown in Figure 6(b): We use three Arduino boards to emulate ECUs in a real car. Another ECU processor with Linux operating system runs AutoE2E and connects to these Arduino boards with USB cables. AutoE2E is implemented in C++.

1) ECU Configuration: We use three Arduino boards to emulate the ECU processors. Based on the task configurations in [31] that an independent micro-processor is deployed for each actuator, we use two Arduino boards to control the yaw angle and the speed for the steering and motor systems, respectively. Another Arduino board is used as the computation unit for control-based calculations. To enable communications between the Arduino boards, we use jumpers to connect these devices through serial communication ports. FreeRTOS is installed as the real-time operation system to schedule the tasks on the scaled car, which is a lightweight real-time kernel which can provide features like priority assignment and preemptive scheduling, and is sufficient for our testbed implementation.

2) End-to-End Task Configuration: We set four end-to-end tasks in the testbed as shown in Figure 7: T_{11} emulates the steering-by-wire application and runs on the steering ECU; 2) T_{21} emulates the drive-by-wire application and runs on the motor ECU; 2) T_{31} and T_{32} emulate the steering control application, which spans across the computation and steering ECUs; 4) T_{41} and T_{42} emulate the speed and stability control application, which spans across the computation and motor ECUs. We choose these tasks because they are the minimum requirements for a fully-functional, autonomous-driving scaled car. To schedule these tasks, we have implemented RMS algorithm for every ECU processor based on the default highest priority first scheduler in FreeRTOS. Note that we use RMS as an example. In a real implementation, AutoE2E can work with other scheduling schemes as long as the corresponding utilization bound can be provided.

3) Deadline and Period Assignment: T_3 and T_4 should have longer deadline requirements than T_1 and T_2 to allow the heavy computation load of subtasks T_{31} and T_{41} on ECU3.

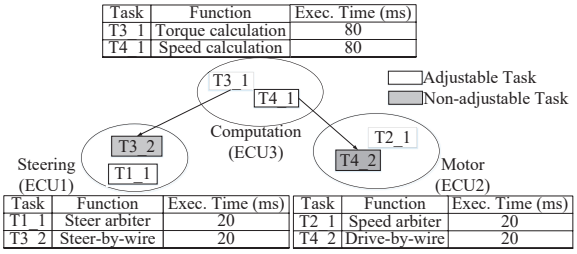


Fig. 7: The workload setup for the AutoE2E testbed.

Thus, because the total execution time of T_3 and T_4 is four times those of T_1 and T_2, we assign the initial deadline of T_3 and T_4 as 200ms, and T_1 and T_2 as 50ms. Note that these deadline requirements can change during the experiment due to speed changes. The end-to-end deadline of each task is evenly divided into subdeadlines for each subtask based on the number of subtasks that the task has, i.e., each subtask's period is $p_i = d_i/n_i$, where n_i is the number of subtasks in task T_i and d_i is the deadline of T_i.

We select the periods of the control loops of AutoE2E as follows: based on the task period, in order to include a sufficient number of subtask instances and minimize the impact of system noise when measuring the ECU processor utilization, we set the control period as 1s for the inner rate-based controller due to the hardware constraints of the Ardunio boards. The control period of the outer precision-based controller is set as 10 times the inner-loop control period to guarantee the convergence of the inner-loop controller based on the analysis in [27]. Note that a much smaller control period can be adopted in the real vehicle because the timeslot of the real ECU processor can be more fine-grained than that of the Ardunio board.

B. Performance of the Outer Precision-based Controller

Here we evaluate the performance of the outer precision-based controller in AutoE2E for different vehicle speed scenarios. We first test how AutoE2E solves the saturation caused by an acceleration process, and then test the performance of the computation precision restorer in the deceleration scenario.

First, we conduct an experiment for the acceleration scenario common to the real automobiles, in which the determined task rate for each control application (i.e., T_1 to T_4) increases several times in this experiment. We compare AutoE2E with EUCON [10], which is designed for general DRE systems. Though EUCON can handle unexpected variations of execution time at runtime, it is still impossible to let the ECU processor utilization converge to the utilization bound for some stringent task rate requirements, especially in the high speed case. One advantage of AutoE2E over EUCON is that AutoE2E uses an outer precision-base controller to solve the saturation issue without suffering deadline misses, and tries to maintain the computation precision as high as possible.

Figure 8 shows the ECU processor utilizations, computation precision and deadline miss ratio for AutoE2E and EUCON. Figure 8(a) shows that EUCON cannot find a suitable task rate above the determined task rates to control the ECU processor utilizations: the utilizations of ECU1 and ECU2 stay above the utilization bounds after 100s, and the utilizations of all

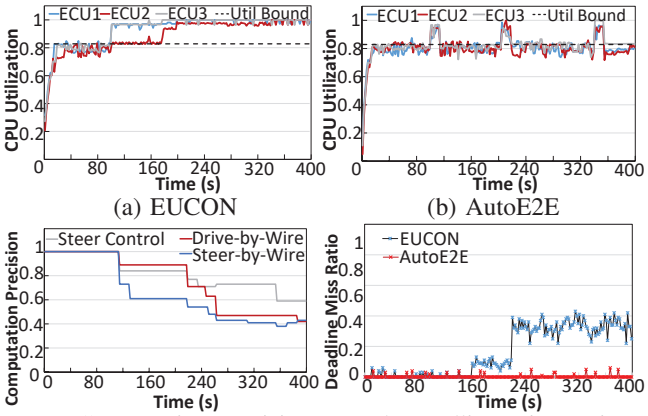


Fig. 8: The comparison between EUCON and AutoE2E with an acceleration process, where the determined task rates become higher at 100s, 200s, and 320s.

ECUs become nearly one after 200s, indicating more deadline misses. In contrast, Figure 8(b) shows that AutoE2E can solve rate saturation effectively. Though there are some short intervals for the ECU processor utilizations staying above the bound near the changing time point, AutoE2E can quickly decrease the computation precision for some subtasks to maintain the feasibility of the inner rate-based controller. The computation precision of AutoE2E is shown in Figure 8(c), where the tasks experience three large drops of computation precision (at 100s, 200s, and 320s) when the determined task rates become higher, and some small variations due to the uncertainty of the execution time at runtime. Though execution time for some subtasks varies, AutoE2E can still prevent saturation in a close-loop manner. Figure 8(d) shows the deadline miss ratio for the steering control algorithm (T_4). The deadline miss ratio for EUCON is 0.1 at 200s, and increases to 0.45 at 320s. Though there is no continuous deadline miss, these deadline misses can still lead to a large tracking error in the vehicle control application. To conclude, AutoE2E outperforms EUCON by 35.4% in terms of overall deadline miss ratio with an overall task execution time ratio decrease of 24.3%.

Then we analyze the performance of the computation precision restorer. In this experiment, we continue the acceleration scenario shown above as follows: After running at the highest vehicle speed, the vehicle decelerates to the speed where the system experiences the first acceleration (100s in Figure 8(a)). We compare the restorer with two baselines: 1) Optimal, which is the optimal computation precision based on Equation (5). However, the actual execution time is unknown at runtime so it only serves as a theoretical upper bound. 2) Direct Increase, which increases the execution time ratio towards one directly with a fixed step size, and stops once the system is saturated.

Figure 9 shows the performance of the computation precision restorer during the deceleration. Figure 9(a) shows two valleys at 40s and 70s, indicating that the task rate is decreased and the outer-loop controller increases the execution

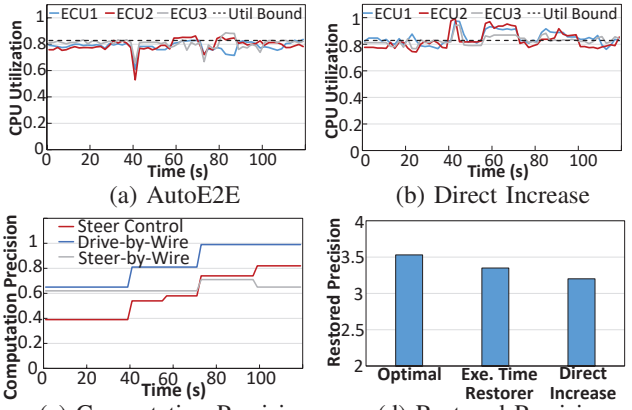


Fig. 9: The performance of the computation precision restorer. The restorer outperforms Direct Increase and is closer to the optimal in terms of computation precision restoration.

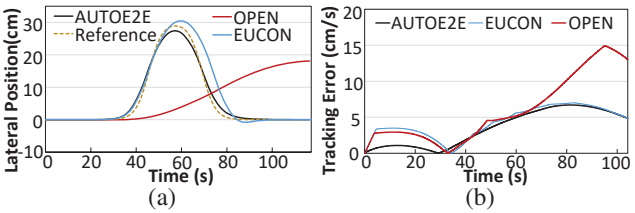


Fig. 10: The performance of AutoE2E compared with EUCON and OPEN: (a) Traveling trajectory for the steering control for a double lane change example. (b) The tracking error for the speed control task in the adaptive cruise control.

time ratio for subtasks. After 80s, the utilizations of ECU3 and ECU2 become higher than the bound, indicating that the current execution time is so large that the inner-loop controller is already saturated. The restorer then stops decreasing the task rates and waits for the outer-loop controller to solve the saturation issue. After the outer outer-loop controller decreases the execution time ratio, the restoration procedure finishes. Direct Increase shown in Figure 9(b) incurs several peaks, indicating that potential deadline misses could happen during the restoration process. In contrast, our restorer leverages under-utilization and does not have such peaks when adjusting the task rate. Figure 9(c) shows how the computation precision is restored during this process. The small variations of the precision can also be found for the steering control caused by the uncertainty of execution time. Figure 9(d) shows that the restored computation precision for our scheme is only 7.7% less than the optimal value (3.35 for the restorer and 3.63 for optimal), and outperform Direct Increase by 8.2%.

C. Comparison of Control Performance

Here we show the performance of AutoE2E with metrics used in the autonomous driving applications. Specifically, we analyze how deadline miss ratios can affect the performance of the real control applications such as lane change path tracking and adaptive cruise control in the scaled car. Figure 10 shows the performance of AutoE2E for the selected autonomous driving applications compared with two baselines: OPEN and EUCON. OPEN is similar to the state-of-the-practice solution used in today's automobile industry, where task rates r are

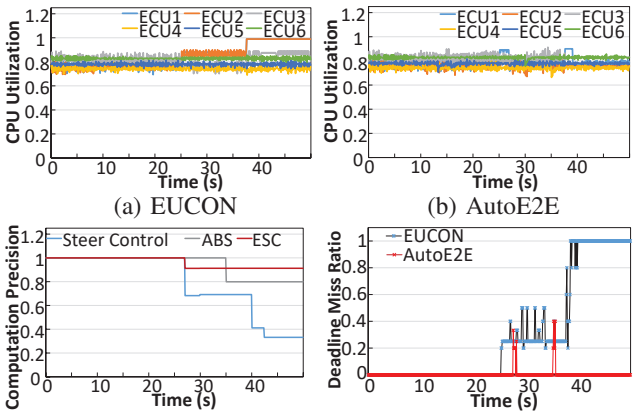


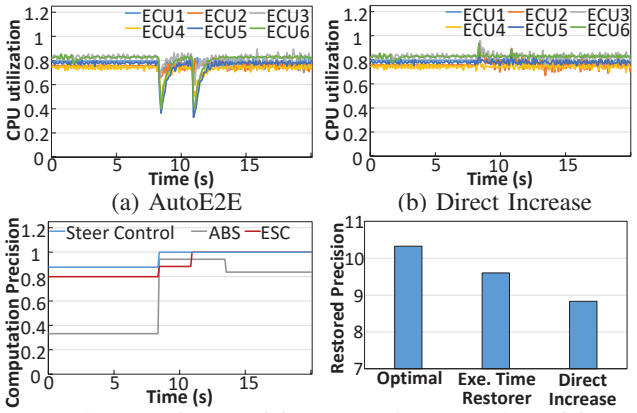
Fig. 11: The comparison between EUCON and AutoE2E with the acceleration process in simulation.

calculated with $\mathbf{F}r = \mathbf{B}$ based on the offline estimated execution time $\mathbf{F}$. Figure 10(a) shows the tracking performance of lane change using steering control (T_4). After 40s, the curve of OPEN shows clear deviation from the reference, which is caused by the continuous deadline misses. EUCON cannot track the trajectory well either, because rate saturation can happen during the experiment, and cause deadline misses for the control applications. AutoE2E, on the contrary, can track the reference with a maximum tracking error of 5cm with a 24.3% decrease of computation precision for the application. AutoE2E outperforms EUCON by 12cm and 5cm in terms of maximum and average tracking error for an 1:16 scaled car, respectively. *Note that the corresponding tracking error difference of a real vehicle is 1.92m and 0.8m, which is sufficiently large for the vehicle to travel across its lane and cause an accident.* Figure 10(b) shows the tracking error for the adaptive cruise control of a high speed vehicle. We can see the spikes in EUCON, which are caused by the abrupt changes in the vehicle dynamics to correct the accumulated error due to the previous deadline misses. Though the tracking error of EUCON is small, these spikes can be harmful to vehicle mechanical parts and decrease the life of the vehicle [32].

D. Larger-Scale Vehicle Simulation Result

In this part, we have implemented AutoE2E in an extended version of the EUCON simulator [10] to show how AutoE2E can handle larger-scale and more realistic autonomous driving control applications with 6 ECUs and 11 tasks. The ECU and task setup is shown in Figure 2.

Figure 11 shows the effectiveness of AutoE2E in terms of saturation prevention. At 25s and 37s, the determined task rates become higher due to the speed increase, which causes the ECU processor utilizations of EUCON to stay above the utilization bounds in Figure 11(a). The corresponding deadline miss ratio increases to 1 after 37s as shown in Figure 11(d). On the contrary, AutoE2E can control the ECU processor utilization to the utilization bound by adjusting both the execution time ratios and the task rates, and only has two short intervals of staying above the utilization bounds at 25s and 37s as shown in Figure 11(b). After that, AutoE2E decreases the execution time ratio for some control applications and



(a) AutoE2E (b) Direct Increase

(c) Computation Precision (d) Restored Precision

Fig. 12: The comparison of the computation precision restorer and the baselines for an emulated deceleration scenario.

guarantees the feasibility for the inner-loop controller as shown in Figure 11(c). To conclude, AutoE2E can still guarantee the deadlines for these applications for the heavy workload.

We then test the performance of the computation precision restorer. Figure 12 shows the utilization valleys when we apply the computation precision restorer. After two rounds of task rate decrease, almost all the computation precision can be restored to the original value. On the other side, Direct Increase shows spikes around 8s, which could lead to possible deadline miss. Figure 12(d) shows that our scheme can achieve a computation precision of 9.62, which is 3.9% lower than optimal, and outperforms Direct Increase by 12.9%.

VI. Conclusion

The recent growth of autonomous driving has introduced a new research challenge on real-time scheduling of vehicle control systems. After examining traditional open-loop scheduling methodologies used in automotive systems and adaptive real-time scheduling proposed for general DRE systems, we find that a new real-time scheduling solution must be designed due to a special feature of driving control. In this paper, we have proposed AutoE2E, a two-tier middleware system for automotive OS that overcomes the limitation of existing solutions by having a second-tier controller, to dynamically lower the execution time (and so computation precision) within the allowed range to regain effective CPU utilization control for E2E real-time guarantees. AutoE2E has been evaluated both on a hardware testbed with scaled cars and in larger-scale simulation. Our results demonstrate that AutoE2E outperforms a state-of-the-art solution that relies solely on rate adaptation by 35.4%, on average, in terms of deadline miss ratio, and in the meantime reduces the tracking error of path tracking.

References

[1] "Tesla's autopilot driver-assistance system." [Online]. Available: https://www.tesla.com/autopilot

[2] "Volvo's pilot assist." [Online]. Available: http://volvo.custhelp.com/app/answers/detail/a_id/9731/~/pilot-assist

[3] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer*, 2009.

[4] W. Chang, S. Chakraborty *et al.*, "Resource-aware automotive control systems design: A cyber-physical systems approach," *Foundations and Trends in Electronic Design Automation*, 2016.

[5] "Explanation of application in-terfaces of the chassis domain." [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_AIChassis.pdf

[6] A. Girault, C. Prévot, S. Quinton, R. Henia, and N. Sordon, "Improving and estimating the precision of bounds on the worst-case latency of task chains," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[7] A. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ecu networks: modeling and precise estimation," in *EMSOFT*, 2010.

[8] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *RTSS*, 1998.

[9] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *OSDI*, 1999.

[10] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, 2005.

[11] ——, "End-to-end utilization control in distributed real-time systems," in *ICDCS*, 2004.

[12] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, 2007.

[13] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, 1973.

[14] G. C. Buttazzo, E. Bini, and D. Buttle, "Rate-adaptive tasks: Model, analysis, and design issues," in *DATE*, 2014.

[15] X. Wang, Y. Chen, C. Lu, and X. D. Koutsoukos, "Towards controllable distributed real-time systems with feasible utilization control," *IEEE Transactions on Computers*, 2009.

[16] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *DAC*, 2007.

[17] Y. Chen, C. Lu, and X. Koutsoukos, "Optimal discrete rate adaptation for distributed real-time systems," in *RTSS*, 2007.

[18] L. Greco, D. Fontanelli, and A. Bicchi, "Design and stability analysis for anytime control via stochastic scheduling," *IEEE Transactions on Automatic Control*, 2010.

[19] M. Chen, C. Nolan, X. Wang, S. Adhikari, F. Li, and H. Qi, "Hierarchical utilization control for real-time and resilient power grid," in *ECRTS*, 2009.

[20] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, 2017.

[21] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich, "Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies," in *RTNS*, 2018.

[22] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*, 2001.

[23] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A framework for end-to-end path delay calculation of automotive systems under different path semantics," in *RTSS*, 2009.

[24] Z. Wang, Y. Bai, J. Wang, and X. Wang, "Parameter selection of an ltv-mpc controller for vehicle path tracking considering cpu computational load," in *ASME*, 2018.

[25] J. E. Kim, O. Rogalla, S. Kramer, and A. Hamann, "Extracting, specifying and predicting software system properties in component based real-time embedded software development," in *ICSE*, 2009.

[26] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[27] X. Wang, X. Fu, X. Liu, and Z. Gu, "Power-aware cpu utilization control for distributed systems," in *RTAS*, 2009.

[28] M. T. Goodrich and R. Tamassia, *Algorithm design: foundation, analysis and internet examples*. John Wiley & Sons, 2006.

[29] "AutoE2E technical report," Tech. Rep., 2020. [Online]. Available: https://www.dropbox.com/s/f0muiqce4730k2u/autoE2E_tech.pdf?dl=0

[30] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, 2007.

[31] R. Verma, D. Del Vecchio, and H. K. Fathy, "Development of a scaled vehicle with longitudinal dynamics of an hmmwv for an its testbed," *IEEE/ASME Transactions on Mechatronics*, 2008.

[32] V. Utkin and H. Lee, "Chattering problem in sliding mode control systems," in *VSS.*, 2006.