# Cloud-assisted Control of Ground Vehicles using Adaptive Computation Offloading Techniques

Arun Adiththan*, Ramesh S† and Soheil Samii†‡
*The City University of New York, New York, NY 10019
†General Motors R&D, Warren, MI 48090
‡Linköping University, 581 83 Linköping, Sweden
Email: arunadiththan@gmail.com, ramesh.s@gm.com, soheil.samii@gm.com

***Abstract*— The existing approaches to design efficient safety-critical control applications is constrained by limited in-vehicle sensing and computational capabilities. In the context of automated driving, we argue that there is a need to leverage resources *"out-of-the-vehicle"* to meet the sensing and powerful processing requirements of sophisticated algorithms (e.g., deep neural networks). To realize the need, a suitable computation offloading technique that meets the vehicle safety and stability requirements, even in the presence of unreliable communication network, has to be identified. In this work, we propose an adaptive offloading technique for control computations into the cloud. The proposed approach considers both current network conditions and control application requirements to determine the feasibility of leveraging remote computation and storage resources. As a case study, we describe a cloud-based path following controller application that leverages crowdsensed data for path planning.**

## I. Introduction

Modern cars increasingly rely on advances in information technology to offer safety-critical features to the users. Advanced driver-assistance systems (ADAS) guide drivers and reduce accidents through various automated features such as adaptive cruise control (ACC), autonomous emergency braking, and forward-collision warning. Such intelligent safety-critical aspects in vehicles are realized by running complex algorithms that process data obtained from several types of sensors such as RADAR, LIDAR, video camera, GPS, etc. The vehicles have limitations such as sensing range and amount of energy required to perform compute-intensive tasks. The in-vehicle capabilities cannot be enhanced through additional hardware (e.g., sensor redundancy) beyond a certain limit without increasing the system complexity and overall cost. We posit that cloud computing technology – that enables on-demand acquistion and release of computational and storage resources – can be used to address some of the existing limitations in terms of sensing and computation.

We view cloud as a technology that not only offers an enhanced computational platform but also as an enriched information source (through means such as crowdsourcing). The enriched information essentially enhances the perception range of a vehicle by fusing together data obtained from multiple vehicles and roadside sensing units. In the context of automated driving, we envisage more and more control applications involving cloud being developed in the future. While there are certain benefits for leveraging resources "out-of-the-vehicle", the network bandwidth supported by existing communication standards is not adequate to meet hard real-time constraints of core control computations on a large scale. Therefore, it is essential to explore various parts of control application computation amenable for offloading to the cloud. For example, the fault assessment of vehicle state estimation algorithms operate at a large sampling interval (compared to the main control loop in a vehicle stability control module) and hence can be a good candidate for offloading to the cloud. It is also necessary to develop techniques required to deal with potential failure scenarios (such as transient connectivity loss and large delays) so that the safety and stability of the control system are not compromised.

In this light, we propose an adaptive offloading controller architecture that determines when it is feasible and beneficial to offload control computations, that may leverage additional data and computational resources, to the cloud. The proposed offloading approach is opportunistic and hence it is flexible to toggle between in-vehicle and cloud-based resources. We demonstrate the feasibility of our approach using a cloud-based path following controller case study in Matlab. Initial results show an improved controller performance in terms of both distance traveled by the vehicle and the time taken between a given source and destination, even in the presence of failures in the communication network.

## II. Cloud-assisted computation offloading architecture

The proliferation of ADAS features and the push for autonomous vehicles are leading to extra computational and sensory capabilities involving array of sensors which bring in large volume of data. Utilizing purely in-vehicle infrastructure is neither scalable nor cost-effective. Therefore, we explore the possibility of leveraging "out-of-the vehicle" resources based on edge and cloud technology which are becoming common these days. In this section, we discuss the problem of computation offloading and describe a cloud-assisted adaptive computation offloading technique for automotive applications.

### A. Computation Offloading

The objective of computation offloading is to improve performance by leveraging powerful processors, enable advanced functionality, save costs, and preserve limited resources such as energy in a client device. The computation offloading may

978-3-9819263-0-9/DATE18/©2018 EDAA

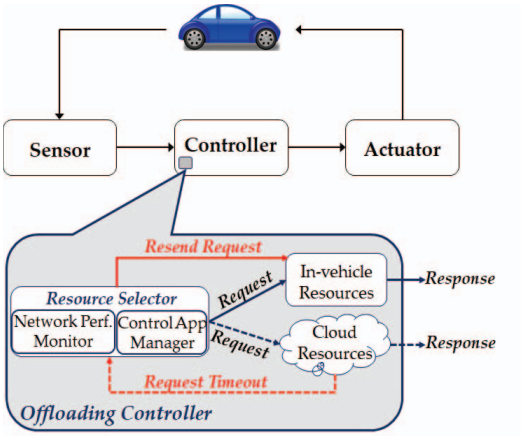


Fig. 1: Computation Offloading Controller Architecture

be performed at different granularity levels ranging from code and tasks to entire applications. Several aspects of the computation offloading problem [1] and cloud-based augmentation techniques [2] have been studied in the mobile cloud computing domain. For mobile devices, major emphasis is placed on saving energy and hence the computation offloading decisions are largely determined by the viability of network conditions at any given point of time. In the automotive domain, however, the most important consideration is satisfying the real-time constraints of the control applications so that the safety and stability of the vehicle is ensured. A typical control loop in an application like adaptive cruise control (ACC) takes 10 ms. This, given the current network communication standards, makes it infeasible to offload core control computation to the cloud. Therefore, we need to identify parts of control computation amenable for offloading to the cloud without compromising on the safety and stability of the system. With the existing communication technology, the candidate computational part would ideally have a delay tolerance in the range of 150-200 ms and potentially benefit from additional computation resource and/or data accessed via the cloud.

### *B. Adaptive Offloading Controller Architecture*

Figure 1 shows a high level view of our offloading controller architecture for control applications. The basic functionality of the offloading controller is to determine whether to execute a given computational task on a local or remote resource. The decision-making process relies on two components, namely, Network Performance Monitor and Control Application Specification. The network performance monitor module continuously monitors the network characteristics such as available bandwidth and uplink data rate. The control application specification component identifies the self-contained parts of control computation (e.g., pre-processing of sensor data, vehicle state estimation) and specifies their data transfer, computation, and latency requirements. The task specification along with the current network conditions guide the process of selecting a resource (local or cloud). Once a suitable resource for a given task is identified, the task is executed on the resource and the response is fed to the control loop.

**Algorithm 1:** Task Offloading Controller

```
/* Computation task T_i = (T_i^in, S_i, C_i, L_i) where,        */
/*      T_i^in: task input                                      */
/*      S_i, S_i': size of the input & output data              */
/*      C_i: # of CPU cycles required for task T_i              */
/*      L_i: latency requirement of the task T_i                */
/* C_cloud: # of CPU cycles available in the cloud              */
/* β, β': available uplink and downlink data rate              */
/* FLAG_off: set to 1, if offloading is feasible               */

Input  : Set of input tasks T = {T_i}, i = 1, 2, ..., n
Output: Task execution output: T_i^out

FLAG_off:=0
while T ≠ NULL do
  measure available uplink data rate for task T_i: β
  estimate network delay: δ_i^net := S_i/β + S_i'/β'
  estimate computation delay: δ_i^comp := C_i/C_cloud
  if δ_i^net+δ_i^comp < L_i then
    FLAG_off:=1 // feasible to offload to cloud
  end if
  if FLAG_off==1 then
    start timer
    transfer task T_i to the cloud
    while not_timeout do
      wait-for T_i^out from the cloud
    end while
    if timeout then
      FLAG_off:=0 // redirect task to onboard resources
    end if
  else
    T_i^out = execute task onboard
  end if
  return T_i^out
end while
```

Algorithm 1 describes the step-by-step process of resource selection and offloading. Given a task description, the first step is to assess the current network parameters, i.e., measuring the available uplink data rate for input data transfer. Based on the available data rate, we compute the network delay. Using both network delay and computation latency, we then determine the feasibility of offloading a specific task to the cloud[1]. If a given task is deemed feasible for offloading, we initiate the data transfer process and start the timer. The offloading controller waits until timeout or task output is returned. In the case of timeout[2], the task must be redirected to the onboard resource for execution. This step is important to ensure continued safety and stability of the vehicle.

Algorithm 2 shows the steps involved in the remote server once a task is offloaded to the cloud. The incoming task

[1]We assume that the code/procedure to execute a specific task is already available on the cloud server. It is also possible to transfer a custom code during the task offloading, if the available bandwidth is sufficient.

[2]The task execution request sent to the cloud might timeout due to various reasons. E.g., request/response packet loss, a sudden spike in congestion, transient resource outage in the cloud infrastructure, etc.

**Algorithm 2:** Task execution in the cloud

**Input :** Input task queue $\mathcal{T}_{queue}$; Cloud datastore (CDS)
**Output:** Task execution output: $T^{out}$

**enqueue** task request into $\mathcal{T}_{queue}$
**while** $\mathcal{T}_{queue}$ not EMPTY **do**
  **select** task $T_j \in \mathcal{T}_{queue}$
  **if** Task $T_j$ requires additional data from CDS **then**
    $T_j^{inCDS}$ := **query** $T_j$ relevant data from CDS
    $T_j^{in}$:=FUSE($T_j^{in}$, $T_j^{inCDS}$) // data fusion
  **end if**
  **select** a server instance, $serv_{id}$, for task $T_j$
  $T_j^{out}$ = **execute** task $T_j$ on $serv_{id}$
  **return** $T_j^{out}$
**end while**

requests are added to the task queue. Depending on the application requirements, in addition to the enhanced processing capability, the data available via the cloud datastore can be leveraged. The enriched data input for the task, which may improve the overall control quality by enhancing the perception range of the vehicle, is achieved by fusing together the task execution input data sent by the vehicle and the data retrieved from the cloud datastore. The task then gets assigned to a server instance and the output is returned to the vehicle.

## III. Case Study: Cloud-based Path Following Controller

We have utilized the NS-3 LTE library [3] for modeling the communication channel shared by multiple vehicles. The latency between the vehicle and remote cloud server for several network and vehicle configurations are obtained and used in a Matlab model of path following controller.

### A. Communication channel modeling using NS-3 LTE library

The simulation setup in NS-3 includes one base station (aka eNB node). We used constant velocity mobility model for the vehicles (45 mph). The downlink and uplink frequency bands of the cell are set to 2110 and 1710 MHz, respectively. A Proportional Fair MAC Scheduler and Friis path loss model are used in the communication link between the vehicles and base station. The transmission method between the eNB node and vehicles is a single-hop unicast. Two distinct types of applications are configured on each remote host – a UDP Echo application on Remote Host 1 and a packet generator application on Remote Host 2 (corresponding to vehicular control application and video streaming traffic). The control application traffic constitutes 80% of the total traffic and the remaining 20% is the entertainment traffic (1024 B packet every 20 ms). The packet size of the control application traffic transmitted by all but one vehicles is in the range of 128 to 2048 B every 20 ms. The remaining one vehicle, for which we are measuring the latency values, transmits 512 B packet every 20 ms. Table I shows the latency values obtained by varying number of vehicles and the network bandwidth. The latency values noted is an average of 10 runs.

| Bandwidth \ # vehicles | 15 | 30 | 45 | 60 | 75 |
|---|---|---|---|---|---|
| 5 MHz | 51.44 | 52.09 | 52.92 | 99.49 | 100.66 |
| 10 MHz | 37.28 | 42.01 | 42.1 | 68.49 | 89.38 |

TABLE I: Data transfer latency (ms)

### B. Path following controller simulation in MATLAB

We had chosen the pure pursuit controller algorithm that is available in the Robotic Systems Toolbox in Matlab [4]. The sampling interval of path planning component for the pure pursuit controller meets the offloading constraints specified in Section II and hence identified as a suitable candidate to be executed in the cloud.

The pure pursuit controller implements a path tracking algorithm that keeps track of and controls the robot's movements along the path. The path between a source and destination comprised of a set of waypoints (or intermediate (x,y) coordinates). The fundamental operation in the controller is the computation of linear and angular velocities for the next control step based on the current pose of the robot. The posespecified as [x,y,theta], where (x,y) represents a point in the reference coordinate system and "theta" represents the heading angle of the robot.

In this work, we have extended the static waypoint generation and path following controller code (available at [4]) to dynamically generate waypoints using purely on-board or remote cloud resources. Also, we demonstrate the switching between local and cloud resources under failure scenarios such as network connectivity loss and large delays. The cloud-based waypoint generation factors in the most up-to-date information about the constantly changing road conditions and may compute a more efficient path for the robot.

We adopt a primitive approach for waypoint generation. We assume that the robot is operating in an environment where from any given point, there are three potential directions – namely, vertical, horizontal, or diagonal – the robot can move on. Each potential direction has a weight associated with it that captures the "quality" of the path (e.g., if a path has a pothole or stalled traffic, it will have less weight and hence considered as a low-quality path). The primitive waypoint generation algorithm always picks the path with the highest quality. In the local-only waypoint generation approach, the weights either remain static or change at a much lower frequency compared to the cloud.

Figures 2(a) & 2(b) show the comparison in performance metrics – distance traveled and time taken – between a purely in-vehicle and cloud-based path planning approaches. In the later case, the changes in road conditions are factored in using crowdsourced data obtained from the cloud server. In the cloud-based waypoint generation case, we used the communication latency values obtained for N=30 vehicle case in the NS-3 simulation. Figures 2(c) & 2(d) show path planning under communication network failure scenarios. When a failure is detected via Network Performance Monitor component of

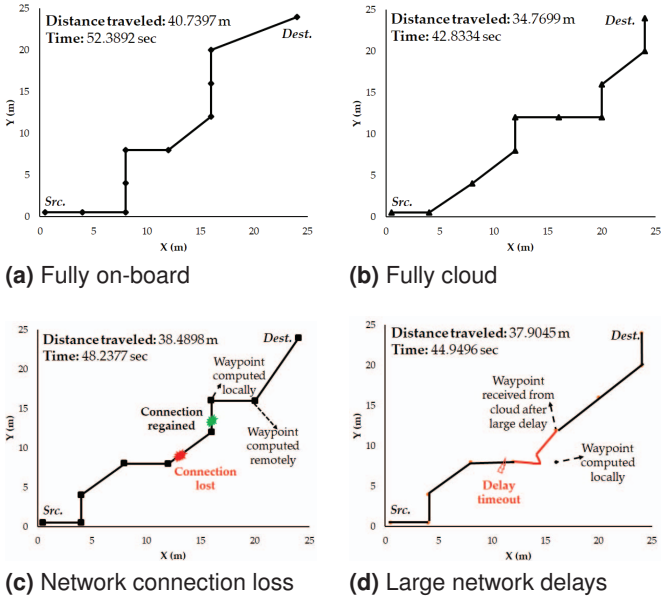


Fig. 2: Waypoint generation for a path following controller – in-vehicle vs remote computation scenarios

the offloading controller (c.f. Figure 1), the resource selector component chooses to leverage local resources for the next waypoint computation. Since the local knowledge is either static (or less frequently updated) compared to the cloud, it results in a less efficient path than a purely cloud-based approach. In case 2(d), once the next waypoint from the cloud arrives after a large delay, the robotic controller would update its intermediate goal and starts moving towards that direction.

## IV. Related Work

A cloud-assisted approach for obstacle detection and avoidance for autonomous vehicles is proposed in [5]. The system combines in-vehicle and roadside sensor data (accessed via cloud) to detect obstacles ahead of time. The results reported in this work show a better reaction time to obstacles compared to a non-cloud-based collision avoidance system. Motion planning involves computing dynamically feasible trajectory for a given source and destination pair. Different approaches to motion planning for self-driving cars and their computational complexity is discussed in [6].

A declarative approach to detect dangerous events by combining vehicle sensor data and static road information obtained from a cloud database is proposed in [7]. The declarative rules are defined using a sensor predicate and a cloud predicate. A cloud-based vehicle location tracking approach for urban environments, called CARLOC, is proposed in [8]. A probabilistic position that uses odometer, vehicle heading direction information from the vehicle and a cloud-hosted online map database and landmark information is used to achieve lane level accuracy in determining the location of a vehicle.

A software architecture for cloud-based automotive control is proposed in [9]. The proposed futuristic approach involve only sense and actuate functions in the physical domain (i.e., in-vehicle) whereas the control computations are performed entirely in the cloud. The Control as a Service (CaaS) model allows vehicle owners to access advanced control features on-demand. An architecture for cloud-based remote vehicle diagnostics, called AutoPlug, is proposed in [10]. The proposed architecture would enable proactively detecting software faults using diagnostic trouble codes, and allow code updates to be dispatched from a remote datacenter to vehicle electronic control units.

In contrast with the works cited above, our work aims to arrive at a generic approach to leverage cloud for control applications based on computational resource/data availability. Our approach also factors in varying communication link conditions (such as path loss and large delays). In the event of a communication failure, stability of the vehicle is ensured by dynamically switching to in-vehicle computational resources.

## V. Conclusion

Critical automotive control applications can leverage remote computing technologies (such as cloud) to enhance in-vehicle and fault tolerance capabilities. In this work, we presented a novel adaptive offloading controller architecture that would enable the realization of remote resource based control applications. We had demonstrated the feasibility of our communication failure-tolerant offloading approach using a cloud-based path following controller applciation. In future, this work can be extended to study a more comprehensive control application and quality metrics that would allow us to perform a detailed trade-off analysis between the fully local and remote computation approaches.

## References

[1] K. Kumar, J. Liu, Y. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[3] NS3-LTE Library. https://www.nsnam.org/docs/models/html/lte.html, 2016. [Online; accessed Aug. 18, 2017].

[4] Path following controller. https://www.mathworks.com/help/robotics/examples/path-following-for-a-differential-drive-robot.html, 2015. [Online; accessed Aug. 18, 2017].

[5] S. Kumar, S. Gollakota, and D. Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 41–46. ACM, 2012.

[6] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[7] Y. Jiang, H. Qiu, M. McCartney, W. Halfond, F. Bai, D. Grimm, and R. Govindan. CARLOG: A platform for flexible and efficient automotive sensing. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 221–235. ACM, 2014.

[8] Y. Jiang, H. Qiu, M. McCartney, G. Sukhatme, M. Gruteser, F. Bai, D. Grimm, and R. Govindan. CARLOC: Precise positioning of automobiles. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 253–265. ACM, 2015.

[9] E. Hasan. Control as a service (CaaS) : Cloud-based software architecture for automotive control applications, 2015.

[10] R. Mangharam. The car and the cloud: Automotive architectures for 2020. *The Bridge*, 2012.