University of Pennsylvania
ScholarlyCommons

Real-Time and Embedded Systems Lab (mLAB) School of Engineering and Applied Science

10-1-2011

AutoPlug: An Automotive Test-bed for Electronic Controller Unit Testing and Verification

Utsav Drolia
University of Pennsylvania, utsav@seas.upenn.edu

Zhenyan Wang
University of Pennsylvania, zhenyan@seas.upenn.edu

Yash Pant
University of Pennsylvania, yashpant@seas.upenn.edu

Rahul Mangharam
University of Pennsylvania, rahulm@seas.upenn.edu

Suggested Citation:
Drolia, U.; Zhenyan Wang; Vemuri, S.; Behl, M.; Mangharam, R.; , "Demo abstract: AutoPlug — An automotive test-bed for ECU testing, validation and verification," Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on , vol., no., pp.131-132, 12-14 April 2011

©2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/mlab_papers/37
For more information, please contact repository@pobox.upenn.edu.

AutoPlug: An Automotive Test-bed for Electronic Controller Unit Testing and Verification

Abstract
In 2010, over 20.3 million vehicles were recalled. Software issues related to automotive controls such as cruise control, anti-lock braking system, traction control and stability control, account for an increasingly large percentage of the overall vehicles recalled. There is a need for new and scalable methods to evaluate automotive controls in a realistic and open setting. We have developed AutoPlug, an automotive Electronic Controller Unit (ECU) test-bed to diagnose, test, update and verify controls software. AutoPlug consists of multiple ECUs interconnected by a CAN bus, a race car driving simulator which behaves as the plant model and a vehicle controls monitor in Matlab. As the ECUs drive the simulated vehicle, the physicsbased simulation provides feedback to the controllers in terms of acceleration, yaw, friction and vehicle stability. This closedloop platform is then used to evaluate multiple vehicle control software modules such as traction, stability and cruise control. With this test-bed we highlight approaches for runtime ECU software diagnosis and testing of the stability and performance of the vehicle. Code updates can be executed via a smart phone so drivers may remotely "patch" their vehicle. This closedloop automotive control test-bed allows the automotive research community to explore the capabilities and challenges of safe and secure remote code updates for vehicle recalls management.

Disciplines
Engineering

Comments
Suggested Citation:
Drolia, U.; Zhenyan Wang; Vemuri, S.; Behl, M.; Mangharam, R.; , "Demo abstract: AutoPlug — An automotive test-bed for ECU testing, validation and verification," Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on , vol., no., pp.131-132, 12-14 April 2011

©2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/mlab_papers/37

AutoPlug: An Automotive Test-bed for Electronic Controller Unit Testing and Verification

Utsav Drolia Zhenyan Wang Yash Pant Rahul Mangharam
Department of Electrical & System Engineering
University of Pennsylvania
{utsav, zhenyan, yashpant, rahulm}@seas.upenn.edu

***Abstract*— In 2010, over 20.3 million vehicles were recalled. Software issues related to automotive controls such as cruise control, anti-lock braking system, traction control and stability control, account for an increasingly large percentage of the overall vehicles recalled. There is a need for new and scalable methods to evaluate automotive controls in a realistic and open setting. We have developed AutoPlug, an automotive Electronic Controller Unit (ECU) test-bed to diagnose, test, update and verify controls software. AutoPlug consists of multiple ECUs interconnected by a CAN bus, a race car driving simulator which behaves as the plant model and a vehicle controls monitor in Matlab. As the ECUs drive the simulated vehicle, the physics-based simulation provides feedback to the controllers in terms of acceleration, yaw, friction and vehicle stability. This closed-loop platform is then used to evaluate multiple vehicle control software modules such as traction, stability and cruise control. With this test-bed we highlight approaches for runtime ECU software diagnosis and testing of the stability and performance of the vehicle. Code updates can be executed via a smart phone so drivers may remotely "patch" their vehicle. This closed-loop automotive control test-bed allows the automotive research community to explore the capabilities and challenges of safe and secure remote code updates for vehicle recalls management.** [1]

I. Introduction

In 2010, safety recalls affected 20.3 million vehicles, according to the National Highway Traffic Safety Administration (NHTSA) [1], [2]. All together since 1966, when NHTSA's record of recalls begins, the industry has recalled more than 470 million vehicles. Over the past decade, software issues in the automotive electronic controller units (ECUs) have begun to account for an increasing larger share of the source of recalls. A typical modern car will include 40-70 ECUs, each controlling a specific function of the vehicle. These ECUs contain over 100 million lines of code [3]. (As a point of reference, Microsofts Windows XP contained about 40 million lines of code, when it was released in 2001 [4].) Consider for example, in a 2009 defect notice filed with the NHTSA, Volvo recalled 17,000 vehicles [5] for the following software-related issue: "The engine cooling fan may stop working due to a software programming error in the fan control module." In the short term, the problem could lead to "reduced air conditioning performance." However, if not corrected, it could lead to "loss of cooling system function and engine failure. The driver may not have sufficient time to react to the warning lights or the text message in the instrument panel, increasing the risk of a crash." While critical recalls require vehicles to be repaired by the dealership, a large proportion of software-related recalls can be remotely upgraded at lower cost.

There is an urgent need for systematic analysis of software bugs in automotive control systems, a system to *remotely* diagnose the software on a set of ECUs, perform safe and secure remote code updates and finally execute methods for online testing and runtime verification for evidence-based confirmation of the safety and efficacy of the code update. Remote vehicle recalls management will initiate network-wide preventive maintenance and deliver a means for continuous on-line performance monitoring and reduce the cost of warranty management.

A. Remote ECU Recalls Management

Software-related issues with automotive ECUs will continue to be a dominant reason for recalls in the future. The cost of servicing these recalls by manually servicing each vehicle significantly impacts the competitiveness of the automotive manufacturer. Furthermore, currently all vehicles of the given make/year/model are recalled while the issue may only affect a small percentage of the vehicles. There is currently no means to distinguish the affected vehicles from those that continue to be safe and efficient. There is, therefore, an urgent need to explore remote warranty management and remote recalls servicing of software. This can be accomplished by interfacing the vehicle's computer to a smartphone and running remote code diagnostics, code updates/patches, testing and safety certification. While this approach may not be applicable to the most critical issues, it will be able to service non-critical and cabin control issues. The approach here involves the following steps (see Fig. 1):

1. Software Bug Detection: When the automotive manufacturer is alerted about a potential software issue in a particular vehicle model, a broadcast with a diagnostic test is sent to the potentially affected vehicle owners. The code is loaded in a secure manner from the owner's smartphone into the vehicle via a WiFi gateway interfacing the vehicle's on-board diagnostics (OBD) computer.

2. Online Diagnostics: The diagnostic code runs a set of invasive and non-invasive tests which determine if the vehicle is to be recalled or not.

3. Remote Code Updates: If a recall is necessary, the owner has the option to conduct the recall remotely by downloading a patch for the ECU software. The patch is

[1]All student authors contributed in equal parts. This research work was supported by NSF CPS-0931239, CSR-0834517 and MRI-0923518 grants.

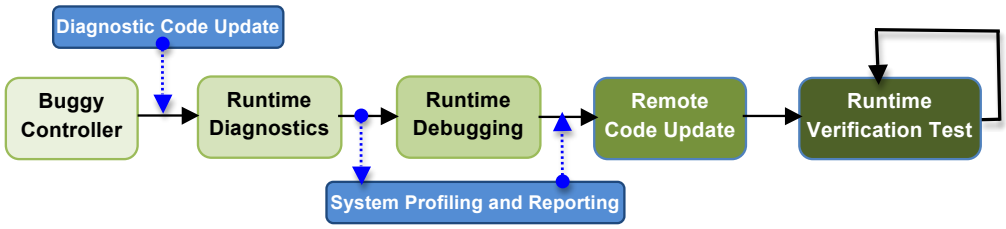

Fig. 1. Procedure for remote ECU software diagnostics, updating and testing

applied automatically to the set of target ECUs.

4. Runtime testing and verification: provide evidence-based proof that the applied patch is safe and does not violate the properties of the affected ECUs.

B. Research Challenges

With the proposed procedure, there are significant challenges with ensuring that the communication and write actions to the vehicle are safe and secure; we focus on safety here. As the onus of maintaining safe software in the vehicle is on the manufacturer, there must be both functional and formal approaches to guarantee the safety of the system before, during and after the software patch. The patch will be applied only when the vehicle's engine is off.

The source of software related bugs/errors could range from incorrect parameter settings, random timing delays in code execution to increased noise from a faulty sensor. New methods of system identification will be necessary to identify, isolate and reproduce the source of the error. These will require compact approaches for logging the state of the controller and a new class of efficient provenance protocols for concurrent operation with the controller.

Finally, a combination of runtime testing and verification will be required to ensure the updated controller is suitable for the particular vehicle. After the upgrade, the vehicle will need to be monitored until a sufficient coverage of the controller's state space has been explored.

II. Related Work

Much work has been done in detection and diagnosis of Network Faults. [6] gives an overview of Fault Detection and Diagnosis (FDD) for In Vehicle Electronic Systems and existing methods for FDD in automotive networks as well as at the component and feature-level with focus on Controller Area Networks. [7] presents a software-based implementation and verification scheme for an Automotive FlexRay network, with emphasis on verifying timing of control signals. While the work of these authors provide a basis for detecting and diagnosing network faults, FDD for automotive software-based controllers is yet to be explored. [8] discusses an Instrument-based Verification approach where automated tests are generated in order to check whether models developed in Matlab/Simulink satisfy a set of requirements.

Our focus is on testing the code on the ECU itself for bugs/faults/errors. We use system identification to observe if the transfer function of the implemented controller has changed due to introduction of bugs in the system. We approach the problem from the controls perspective to quantitatively analyze controller errors with the reference.

Software architectures for automotive systems is an active area of research and practice. The motivation behind creating an automotive ECU test-bed with mechanisms for remote diagnosis and software management is based on the idea of the "car as a platform" [9]. A major effort on creating a standard for automotive software architecture, AUTOSAR, specifies models, tools and analytics for design-time development of automotive software [10]. Existing remote vehicle systems, such as GM's OnStar and Ford's Sync, that perform remote servicing [11] are company-specific and do not perform drivetrain/powertrain firmware upgrades nor allow for direct remote management of the firmware.

III. System Architecture

We describe our early efforts in creating AutoPlug, an Open Automotive ECU Test-bed for the development of new software processes to improve the warranty and recall management of vehicles. This platform will help investigate new automotive architectures for remote vehicle software monitoring, testing and updates for more efficient vehicle controls software management. We have created a physical network of ECUs which implement control algorithms for anti-lock braking system (ABS), traction control, stability control and cruise control. In place of a real vehicle we have used The Open-source Race Car Simulator (TORCS) which is a driving simulator that provides physics-based feedback to the ECU network. Matlab is used to set parameters and analyze the output.

AutoPlug consists of three layers: vehicle dynamics *simulation*, *ECU network*, and *middleware* for runtime software diagnostics, debugging, upgrades and verification. The simulation layer models a real automobile (e.g. 1999-2001 Mitsubishi Lancer EVO VI), which provides a form of validation for the ECU operation. The ECU network consists of several microcontrollers (e.g. Freescale HCS12 each running the nano-RK embedded real-time operating system [12]), each of which performs a specific function (e.g. steering, locking/unlocking doors, changing gears). These microcontrollers are networked together using the industry standard CAN automotive protocol [13]. Some functions of vehicles, such as Anti-Lock Braking System, require several ECUs to work together in order to perform the necessary tasks, thus there is a need for them to communicate. The "middleware" layer consists of a small computer that provides a gateway protocol for vehicle manufacturers to interface with the ECU network. We describe the four components of AutoPlug:

A. Plant Model: Vehicle Dynamics Simulator

The TORCS model simulates car dynamics through physics-based modeling, as well as individual car subsystem components (engine, suspension, steering system). However, code was added to support ECU network functions. This included modeling additional vehicle components, as well as integrating the National Instruments NI-6229-M data acquisition cards into the simulation to provide real-time inputs and outputs. The TORCS simulator provided the following features that we deemed essential to the feasibility of our project (including but not limited to):

1) Realistic tire dynamics based on the Pacejka tire model, an industry-standard empirical tire model
2) Variable road conditions
3) Accessible physics engine outputs for use in ECU network (e.g. yaw rate, wheel speeds, later acceleration)
4) Suspension and differential modeling

B. Electronic Controller Unit Network

The ECU network subsystem consists of individual Freescale MC9S12 micro-controllers that perform specific control functions within the vehicle. Fig. 3 shows a general interface between the ECUs and the Controller Area Network (CAN). Each of the eight ECUs are connected to the network in parallel, with every ECU is able to receive the messages that are passed through the circuit. CAN-HIGH and CAN-LOW are the two rails that make up the CAN bus. All ECUs have built-in CAN transceivers and thus can connect directly to the CAN bus.

IV. Electronic Control Units (ECU)

In this section we briefly describe the automotive controls implemented on the ECU network and show the performance of each. The current platform has over nine ECUs which implement controls including accelerator/break pedals, steering control, anti-lock brake system (ABS), differential (yaw) control, cruise control, cabin comfort, transmission, console and propulsion. Nine micrcontrollers are used, with two as gateways to communicate with the simulator and diagnostics interface.

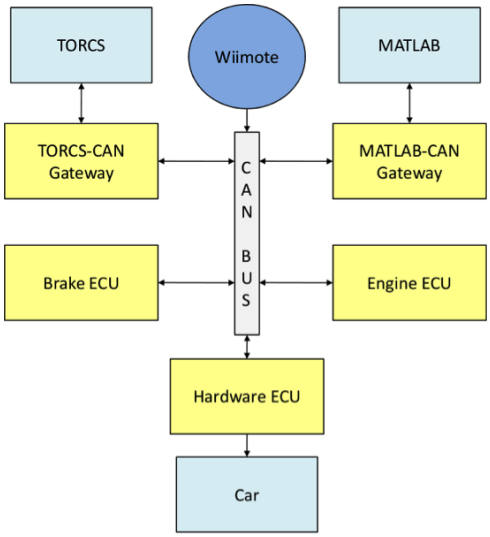

Fig. 2. AutoPlug system architecture

A. Nano-RK embedded RTOS

To address the need for timing precision, priority scheduling and fine-grained resource management, the nano-RK resource kernel [12] has been previously developed with timeliness as a first-class concern. nano-RK is a fully preemptive RTOS with networking support that runs on a variety of embedded network platforms (8-bit Atmel-AVR, 16-bit TI-MSP430 and Freescale HCS12). It supports fixed-priority preemptive scheduling for ensuring that task deadlines are met, along with support for and enforcement of CPU and network bandwidth reservations. Each ECU of AutoPlug test-bed runs nano-RK with a $10\mu s$ OS clock-tick. Each control application is implemented as a task on nano-rk. These tasks can specify their resource demands and the operating system provides timely, guaranteed and controlled access to CPU cycles. In addition, these tasks can be easily activated, suspended or terminated which is quite useful for software diagnostics. This is how we switch between the bugged cruise controller and normal cruise controller, as they are implemented as 2 different tasks. With the RTOS and the ability to program the ECU over CAN, we are able to selectively program each ECU at the task level. This also allows us to load diagnostic tasks at runtime.

B. Stability Control

The stability control is used to keep the car on course and prevent loss of control. It is aided by traction control and ABS in doing this. It measures the current yaw rate and if above a threshold, it signifies that the car is possibly out of control and about to spiral. A message is sent to the Engine Control Unit to reduce the throttle and the brakes are applied according to a proportional controller.

Under no stability control, when the car tries to corner sharp bends at high speeds, hence having a high yaw rate, the wheels tend to skid and the final direction of the car is not in the intended direction. In Fig. 4, we can see that due to hundred percent throttle, the wheels start skidding, sending the car into a whirl, which is signified by the high yaw rate.

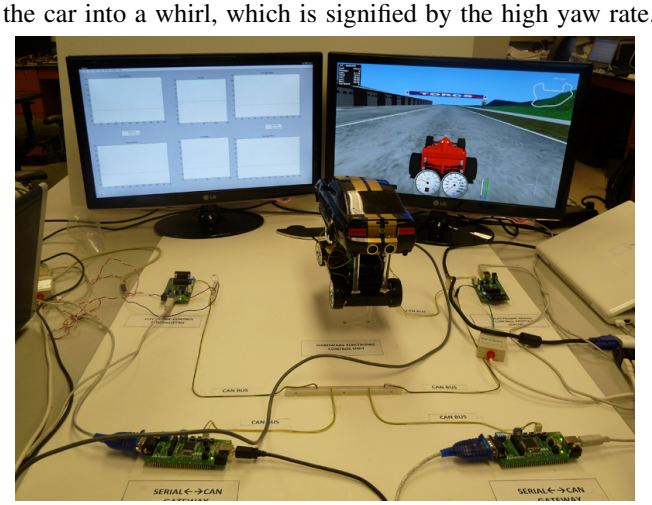

Fig. 3. AutoPlug ECU Test-bed ECUs connected via a CAN bus to the vehicle simulator and system monitor

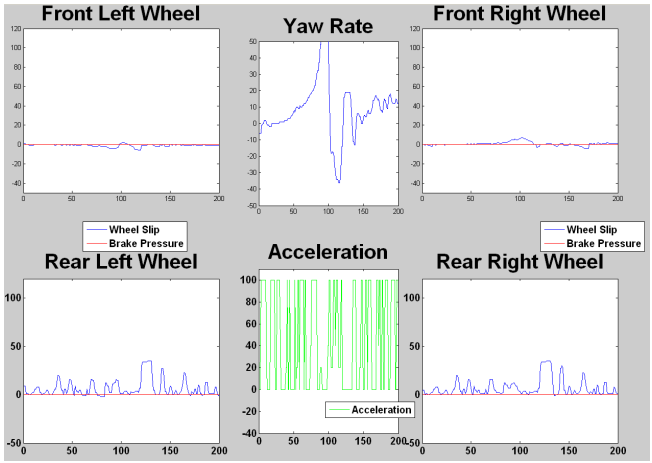

Fig. 4. Stability control OFF

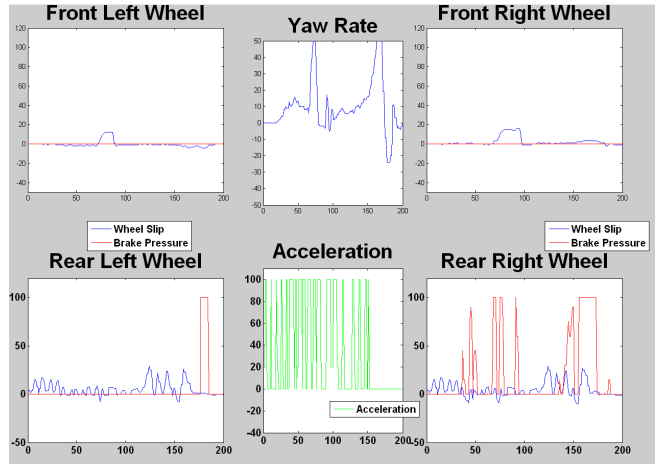

Fig. 5. Stability control ON

Once the the control is switched on, as soon as the car starts to whirl, its speed is immediately reduced and it is almost brought to a halt. Fig. 5 shows that ECU modulates the throttle and the brakes based upon the yaw rate, even if the driver has the throttle completely pressed.

V. Runtime Diagnostics

We now describe on-going and future efforts on diagnostics and verification using the AutoPlug test-bed.

Diagnostic Trouble Codes (DTCs) are codes established by Society of Automotive Engineers (SAE) to report errors in a car over the On-Board Diagnostics (OBD) port in the car. These are predefined codes which cover numerous kinds of errors, and can be generic (applicable to all OBD vehicles) or manufacturer-specific. OBD covers many errors found on electrical and mechanical systems but for the ECU software there are currently only 8 generic codes. Of these, none specify what exactly is wrong with the software. The code indicates that an internal integrity fault was detected if anything within the ECU goes wrong. There are 4 codes for ECU memory malfunction and 1 for programming error. There are no codes for specific software/controls error. If such an error does occur the current approach is to replace the ECU. A less common approach is to re-write the software to the ECU and update the code (i.e. software patch), and that requires taking the car to the dealership. The limited diagnostic information is unable to isolate the exact software faults. Hence, for detecting bugs inside the ECU (the bug might be due to - software, controls implementation, physical errors manifesting themselves in the code) the DTC are insufficient. Even if DTC is extended to test the running code, one cannot be sure of all the possible errors that occur.

We require logs, software probes, breakpoints, traces, a system comparable to a small embedded debugger to certifiably test and analyze running code. Hence, this is a problem of runtime testing and verification of control software and for this what we require more *dynamic* diagnostics and testing. Dynamic Diagnostics and Testing (DDT) will be a method of introducing diagnostic software onto ECUs once a car is observed to be faulty. There would be a library of different diagnostics software for testing, observing, logging different parameters/variables. On observing a car's physical performance, one would be able to determine the possible problems and the kind of error to be detected. This would determine the correct diagnostics test software to be uploaded on to the ECU. Once the software is on the ECU, it will run as a background task, minimally affecting the performance of already running tasks/algorithms. The different kinds of approaches currently being developed for on-line evaluation of control software include:

A. Runtime Verification of results

Runtime verification compares system inputs and outputs to the expected ones. Similar to test case generation, inputs to the system are generated and the corresponding expected output, along with important state variables are stored in a tabular manner. When the run time diagnostics is activated, the diagnostic program would take the input value and search the established table for expected output values, based on the input and current state of other important variables. Then the actual output values will be compared with the expected one. If the deviation between these two values exceeds set threshold, system will alert users that software problems have been detected in the controller and manufacturers should recall the vehicle for more detailed inspection.

B. Control Implementation Verification

This compares the *modeled controller* to the *implemented controller*. We can model the controller using CarSim [11], which produces versatile car and environment models and can be integrated with Simulink. This provides a robust plant model with appropriate controller parameters.

The diagnostic software needs to log inputs, outputs and state variables of the controller software while it is executing. Once there is enough observation data, these logs are extracted from the ECU. Starting with the desired controller parameters (eg: state-space matrices, PID parameters) the logged inputs and state variable values are fed to the modeled controller and the parameters are adjusted with every iteration so that the output matches the logged output. The recorded parameters are compared to the desired

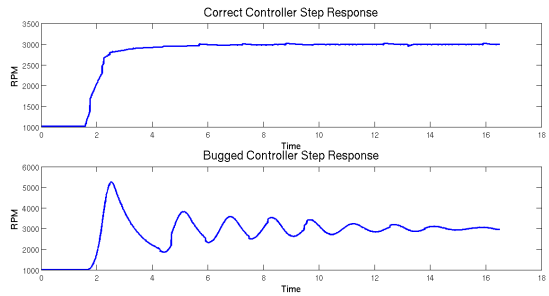

Fig. 6. Comparison between Reference and Bugged Controller

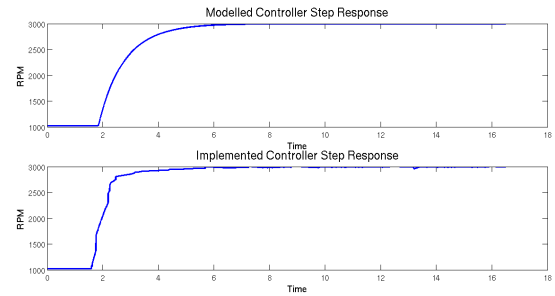

Fig. 7. Comparison between Modeled and Implemented Controller

parameters and the deviations give us information about how the implementation has changed the controller.

C. Sensor Value Validation

This makes sure that the sensors are working correctly. It checks various sensor parameters to detect sensor noise/malfunctions indirectly. To ensure all the sensors report values accurately, related physical values are monitored and compared. For example, if the wheel speed sensor has significant noise then in cruise control achieving the desired cruise speed will be difficult. To make sure that the sensors are working correctly, acceleration sensor values can be used to estimate speed[2] and compared to wheel speed. If these two values deviate from each other significantly, we could say that at least one is faulty.

VI. Controller Verification

The objective is to study the closed loop behavior of a system with both a correct reference implementation and a buggy implementation of the controller. The plant here is the powertrain subsystem of the TORCS car model. We consider the powertrain as the components of the car which results in a rotations per minute (RPM) output at the motor shaft for an acceleration input. An advantage of using the powertrain as a plant is the fact that a step reference is a plausible real world reference signal for a controller to match and is also the most studied signal response in classical control literature.

We propose two methods to differentiate between the behavior of the closed loop system with the reference and buggy controller in the feedback loop and also possibly pinpoint the error in controller gain.

A. Using the step response properties

The most intuitive method of differentiating between two closed loop systems that are supposed to follow the same step reference is to compare the properties of the step response like rise time, peak time, maximum overshoot etc. With knowledge of these parameters it is not very difficult to speculate which of the controller gains (Proportional, Integral or Derivative) is incorrect, but it will never give an exact numerical answer as to how much the gains are. This method is a simple way to invalidate a controller.

To develop a controller, we first modeled a plant in Simulink based on the underlying physics model for the Powertrain in TORCS. The non-linear model thus obtained was used to develop a discrete time PI-controller in C which was imported into Simulink as a S-Function. To validate the controller, we implemented the same C code for the controller on the Engine Control ECU and compared the response to a step input of 3000 RPM obtained from TORCS to that obtained from Simulink. We found a reasonably good match in the step-responses (Fig. 7) and hence considered this controller as our reference controller.

The bugged controller was the same C code but with the values for both gains changed. The same step input of 3000 RPM is given to the controller. The output RPM values logged from the CAN-Bus were imported into MATLAB (Fig. 6). The *stepinfo* command was used to get the rise time, settling time and overshoot with both the reference and bugged controller in the feedback loop (Table I). It is easy to invalidate the bugged controller from both the waveform obtained and the values in the table. It is however not conclusive in pinpointing whether it is just the proportional gain that is wrong or both the proportional and integral gains.

B. Using System Identification

To get a mathematical representation of the powertrain model in TORCS, system identification is done on the testbed with no controller in the closed loop. Binary inputs [14] (with 1 corresponding to full acceleration and 0 corresponding to to acceleration) are sent to TORCS and the RPM output is logged. The Matlab System Identification toolbox is used to get a high order linear approximation of the TORCS

[2]Inertial sensors are not used to estimate speeds in normal cases since they function poorly as speed estimators when there is no acceleration

Controller	Rise Time(ms)	Settling Time(ms)	Overshoot(%)
Reference	1414.8	17005	0.6667
Erroneous	303.8	17723	75.3333

TABLE I
Comparison of Step Response Characteristics

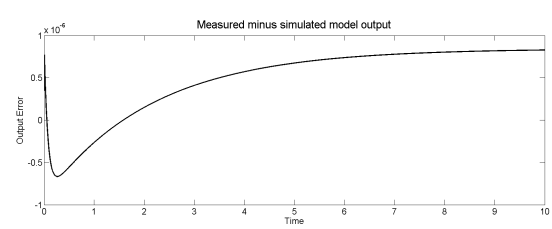

Fig. 8. Error between measured and simulated plant output

powertrain and non-linearities (like saturation and a constant offset) are added to get a very good approximation of the the actual TORCS model. This higher order model with non-linearities is further simplified (through system identification again) to get a third-order linear approximation of the TORCS model (see Fig.8). We aim to get a third-order model as it is generally considered acceptable in most powertrain control literature [15].

The objective here is to identify the closed loop transfer function of the identified plant with a reference controller and then with a buggy controller in the feedback loop. This is also a motivation to get a good linear representation of the TORCS powertrain system. System identification with just a step reference as input to the system is be able to provide closed loop transfer functions which we can decompose to get the gains of both the reference and buggy controller. The PI controller is written in C and is imported into Simulink as a black box (S-Function). The generic transfer function of which is (with K_p and K_i as the proportional and integral controller gains):

$$K_p + \frac{q}{q-1} T_s K_i \tag{1}$$

and can also be written as:

$$C = \frac{(T_s K_i + K_p) - K_p q^{-1}}{1 - q^{-1}} = \frac{n_1 - n_2 q^{-1}}{1 - n_3 q^{-1}} \tag{2}$$

The identified plant is:

$$P = \frac{(2.988q^{-1} + 2.733q^{-2})}{(1 - 1.172q^{-1} + 0.07295q^{-2} + 0.1008q^{-3})} \tag{3}$$

and the closed loop transfer function for the generic PI controller and the plant is:

$$\begin{aligned} \frac{PC}{1+PC} = {} & (2.988n_1q^{-1} - 2.733n_1q^{-2} - 2.988n_2q^{-2} + \\ & 2.733n_2q^{-3})/(1 + (2.988n_1 - n_3 - 1.172)q^{-1} \\ & +(0.07295 + 1.172n_3 - 2.733n_1 - 2.988n_2)q^{-2} \\ & +(2.733n_2 + .1008 - 0.07295n_3)q^{-3} - 0.1008n_3q^{-4}) \end{aligned} \tag{4}$$

By comparing the coefficients of the powers of q in the above transfer function to the identified closed loop transfer function should give us n_1, n_2, and n_3 and hence the gains of the PI controller.

For the closed loop behavior, a constant RPM value (3000 RPM) is taken as the reference input and the output values are logged for both cases. The system identification with these values gives an exact match for both cases with the expected 3rd order numerator and 4th order denominator.

The identified closed loop transfer functions are:

$$CL_{bug} = \frac{(.8967q^{-1} - 1.717q^{-2} + .8198q^{-3})}{(1 - 1.275q^{-1} - 0.4715q^{-2} + 0.8477q^{-3} - .1008q^{-4})} \tag{5}$$

$$CL_{ref} = \frac{(0.2991q^{-1} - 0.5724q^{-2} + 0.2733q^{-3})}{(1 - 1873q^{-1} + 0.6727q^{-2} + 0.3011q^{-3} - 0.1008q^{-4})} \tag{6}$$

Comparing coefficients of q in Eqn. 4 to Eqn. 6 and Eqn. 5 give us $K_p = 0.1$ and $K_i = 0.01$ and $K_p = 0.3$ and $K_i = 0.01$ for the reference and bugged controller respectively (for $T_s = 0.002$) which are exactly the same as the values which we had made the controllers for. This demonstrates a simple method to isolate a controller's buggy software implementation of controller code running on the test-bed. In the near future, we are exploring more techniques for runtime testing and verification of ECU software.

VII. Conclusion

Automotive recalls due to software-related issues are on the rise and will continue to affect a very large number of vehicles. AutoPlug, an open automotive ECU architecture, is proposed for use by the community at large. This will facilitate new and efficient methods to detect, diagnose, update, test and verify automotive ECU software. In this early effort, we presented the initial architecture of the AutoPlug ECU test-bed along with implementations of cruise control, stability control, traction control and anti-lock braking system. The controls are implemented in hardware-based ECUs and interact with a vehicle simulator which provides real-time physics-based feedback. The test-bed is capable in remotely updating code on the ECUs and executing runtime diagnostics and debugging. The test-bed has been used to demonstrate ECU control-software testing and runtime verification to lower the cost of future vehicle recalls. For more details, visit http://www.autoplug.org.

References

[1] Reuters. Ford, GM lead in U.S. auto recalls 2005-2009, 2010.
[2] DailyFinance.com. It's not just Toyota: Auto Recalls Accelerate, 2011.
[3] R. Charette. This Car Runs on Code. Discovery News., 2010.
[4] Economist. Cars and software bugs, 2010.
[5] VolvoOnAut.com. Volvo Recalls over 17,000 Vehicles, 2009.
[6] J. Suwatthikul. *Fault detection*. InTech, 2010.
[7] Wei-Wen Hu, Ming-Li Wang, and Yu-Hui Lin. On the Software-Based Development and Verification of Automotive Control Systems. In *Conference of the IEEE Industrial Electronics Society*. IEEE, 2007.
[8] Arnab Ray, Iris Morschhaeuser, Chris Ackermann, Rance Cleaveland, Charles Shelton, and Chris Martin. Validating Automotive Control Software using Instrumentation-based Verification. In *International Conference on Automated Software Engineering*. IEEE/ACM, 2009.
[9] K. Koscher. Experimental security analysis of a modern automobile. In *Symposium on Security and Privacy*, 2010.
[10] H. Heinecke. Automotive Open System Architecture - An Industry-wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *SAE World Congress*, 2004.
[11] T. Kinjawadekar. Model-based Design of Electronic Stability Control System for Passenger Cars Using CarSim and Matlab-Simulink, 2009.
[12] The nano-RK Sensor Real-Time Operating System.
[13] W. Voss. A Comprehensible Guide to Controller Area Network. *Copperhill Media*, 2005.
[14] Cristian R. Rojas, James S. Welsh, and Graham C. Goodwin. A Receding Horizon Algorithm to Generate Binary Signals with a Prescribed Autocovariance. In *American Control Conference*, 2007.
[15] D. Hrovat and Jing Sun. Models and Control Methodologies for IC Engine Idle Speed Control Design. *Control Eng. Practice, Vol. 5*, 1997.