Technologies for Safe & Efficient Transportation

THE NATIONAL USDOT UNIVERSITY TRANSPORTATION CENTER FOR SAFETY

Carnegie Mellon University | University of Pennsylvania

AUTOPLUG: An Architecture for Remote Electronic Controller Unit Diagnostics in Automotive Systems

FINAL RESEARCH REPORT

Yash Vardhan Pant, Miroslav Pajic, and Rahul Mangharam
University of Pennsylvania

Contract No. DTRT-12-GUTC-11

2012

AUTOPLUG: An Architecture for Remote Electronic Controller Unit Diagnostics in Automotive Systems

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

Miroslav Pajic
University of Pennsylvania, pajic@seas.upenn.edu

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

AUTOPLUG: An Architecture for Remote Electronic Controller Unit Diagnostics in Automotive Systems

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) architecture between the vehicle and a Remote Diagnostics Center to diagnose, test, update and verify controls software. Within the vehicle, we evaluate observerbased runtime diagnostic schemes and introduce a framework for remote management of vehicle recalls. The diagnostics scheme deals with both real-time and non-real time faults, and we introduce a decision function to detect and isolate faults in a system with modeling uncertainties. We also evaluate the applicability of "Opportunistic Diagnostics", where the observerbased diagnostics are scheduled in the ECU's RTOS only when there is slack available in the system. This aperiodic diagnostics scheme performs similar to the standard, periodic diagnostics scheme under reasonable assumptions. This approach works on existing ECUs and does not interfere with current task sets. The overall framework integrates in-vehicle and remote diagnostics and serves to make vehicle recalls management a less reactive and cost-intensive procedure.

Keywords
ECU, Diagnostics, Stability Control, System Identification, Control Systems Diagnostics

Disciplines
Controls and Control Theory | Systems and Communications

AUTOPLUG: An Architecture for Remote Electronic Controller Unit Diagnostics in Automotive Systems

Yash Vardhan Pant Miroslav Pajic Rahul Mangharam
Department of Electrical & System Engineering
University of Pennsylvania
{yashpant, pajic, 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) architecture between the vehicle and a Remote Diagnostics Center to diagnose, test, update and verify controls software. Within the vehicle, we evaluate observer-based runtime diagnostic schemes and introduce a framework for remote management of vehicle recalls. The diagnostics scheme deals with both real-time and non-real time faults, and we introduce a decision function to detect and isolate faults in a system with modeling uncertainties. We also evaluate the applicability of "Opportunistic Diagnostics", where the observer-based diagnostics are scheduled in the ECU's RTOS only when there is slack available in the system. This aperiodic diagnostics scheme performs similar to the standard, periodic diagnostics scheme under reasonable assumptions. This approach works on existing ECUs and does not interfere with current task sets. The overall framework integrates in-vehicle and remote diagnostics and serves to make vehicle recalls management a less reactive and cost-intensive procedure.**

I. INTRODUCTION

The increasing complexity of software in automotive systems has resulted in the recent rise of firmware-related vehicle recalls due to undetected bugs and software faults. In 2009, Volvo recalled 17,614 vehicles due to a software error in the engine cooling fan control module [1]. According to the NHSTA report, the error could result in engine failure and could possibly lead to a crash. In August 2011, Jaguar recalled 17,678 vehicles due to concerns that the Cruise Control in those vehicles may not respond to normal inputs and once engaged could not be switched off [2]. In November 2011, Honda recalled 2.5 million vehicles to update the software that controls their automatic transmissions [3].

While there is a significant effort for automotive software testing and verification at the design stage [4], not all possible runtime faults throughout the vehicle's lifetime can be detected. A systematic approach and infrastructure for post-market runtime diagnostics for the control software is lacking in current automotive systems. Once the vehicle leaves the dealership lot, its performance and operation safety is a blackbox to the manufacturers and the original equipment providers. For the over 100 million lines of code and over 60 Electronic Control Units (ECUs) in a modern vehicle [5], there are only about 8 standard Diagnostic Trouble Codes (DTCs) for software, and those too are generic (e.g. memory corruption)[6]. Out of the DTCs for software, none target the *control software* in the ECUs even though control systems like stability control, cruise control, and traction control are safety-critical systems.

A. Runtime in-vehicle Diagnostics and Recalls Management

The current approach for handling vehicle recalls is reactive where the manufacturers announce a recall only after the problem occurs in a significant population of deployed vehicles and all vehicles of that particular year/make/model are recalled. A software recall involves the vehicle being taken to service center and a technician either manually replaces the ECU which contains the faulty code, or reprograms the code onboard the ECU with the new version provided by the manufacturer. One problem with this method is that the decision to recall vehicles involves word-of-mouth or manually logged information going from the service centers to the manufacturer, which takes time and in the meanwhile may result in a malfunction within a safety critical system. This wait-and-see approach to recalls has a significant cost in both time and money and has a negative impact on the vehicle manufacturers reputation.

The above observations lead us to believe that there is a need for systematic post-market in-vehicle diagnostics for control systems software such that issues can be detected early. The in-vehicle system would be responsible for data logging of sensor values and runtime evaluation of controller states. To complement this, a Remote Diagnostics Center (RDC) would receive this data, over a network link, to prepare an appropriate Fault Detection and Isolation (FDI) response. This would normally be in the form of sending a custom Dynamic Diagnostic Code which observes the ECUs and controller tasks in question. Once sufficient data is captured, the RDC, using a model of the plant is able to execute a grey-box structured system identification to build a plant model of the particular vehicle. Using this vehicle-specific plant model, the RDC develops a fault-tolerant controller for the issue and the vehicle is remotely re-programmed via a code update. While this approach is difficult in theory as it would require extensive runtime verification of the patched controller, we present the early design of such a system with AutoPlug.

B. Overview of AutoPlug

The AutoPlug automotive architecture aims to make the vehicle recalls process a less reactive one with a runtime system for diagnosis of automotive control systems and software. Our focus is on the on-line analysis of the control

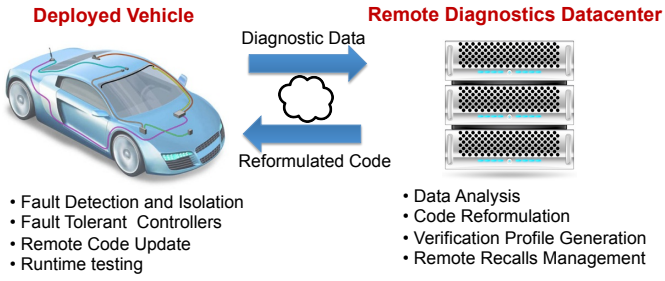

Fig. 1. Overview of the AutoPlug services

system and control software within the vehicle ECU network and between the vehicle and the RDC (see Fig. 1). We assume the network link between the two is available. The runtime system *within the vehicle* is responsible for:

1) **Fault Detection and Isolation** (FDI): Sensor, actuator and controller states are logged for the specific ECU. This data is analyzed locally and a summary of the states are transmitted to the RDC.
2) **Fault Tolerant Controllers**: Once a fault is detected, the high-performance controller is automatically replaced with a backup controller.
3) **ECU re-programming for remote code updates**: Upon reception of updated controller code from the RDC, the runtime system re-flashes the particular controller task(s) with the updated code.
4) **Patched Controller runtime-verification**: The updated code is monitored with continuous checks for safety and performance. We do not focus on this aspect in this paper but consider it in future work.

While the on-board system provides state updates of the specific controller, the *Remote Diagnostics Center (RDC)* provides complementary support by:

1) **Data analysis and fault localization**: Using grey-box structured system identification, a plant model of the particular vehicle is created. The existing controller is evaluated on this model to isolate faulty behavior.
2) **Reformulating Control and Diagnostics Code**: A new controller is formulated for the specific plant model and further diagnostics code is dispatched.
3) **Recalls Management**: Reformulated controller code is transmitted to the vehicle.
4) **Generating Controller Verification profiles**: The updated controller is probed for performance and safety. We do not focus on this aspect in this paper.

A more descriptive view is provided in Fig. 2. Further explanation of these services are presented in Section III.

C. Types of faults

Before the underlying analysis for the diagnostics scheme are explained, it is worth noting the real-time and non-real time faults targeted for detection and isolation in the AutoPlug architecture.

I. Real-Time Faults

Timing issues which are of particular interest to automotive controls are either introduced due to the Controller Area Network (CAN) bus or with the sampling in the sensors/controllers/actuators. These "bugs" are introduced in the hardware-in-loop testbed for stability control, traction control, anti-lock brake system and cruise control and are discussed later.

1) *Delay:* Large delays in transmission of a packet over a network may adversely affect the stability of a closed loop controlled system.
2) *Jitter:* Time varying delay is much more difficult to pinpoint than fixed delays and may also have an adverse effect on the stability and runtime performance of a system.
3) *Incorrect sampling rates:* Different sampling times across the elements of the CAN may result in uncharacteristic behavior of the overall system.

II. System Faults

1) *Stuck-at Faults*: occur when a sensor value stays at the maximum or minimum for a certain period of time.
2) *Calibration Faults:* Calibrating sensors for different environments is a difficult task. Detecting calibration at runtime is important in the context of safety critical control systems.
3) *Noise Faults*: Due to environmental or other electrical reasons, the noise variance in a sensors measurements may be inordinately high and affect sensors/actuators.

The focus of this study will be on sensor faults, especially sensors for lateral displacement, yaw and roll, which play a crucial role in feedback systems for stability and traction control.

The main contributions of this paper are: (a) We present an architecture is introduced which uses both in-vehicle and remote diagnostics for remote recalls management of deployed vehicles; (b) We present a modification of the traditional observer-based FDI scheme for in-vehicle *opportunistic diagnosis*, as well as an experimental thresholding scheme for fault detection and isolation in presence of modeling uncertainties; (c) Finally, we implement and evaluate these schemes on real ECUs on the AutoPlug testbed for Hardware-In-Loop simulation.

D. Organization of the paper

Section III explains the key components of the AutoPlug architecture and their working. A simple example to explain the diagnostics scheme and the underlying math is covered in Section IV. The AutoPlug test-bed which is used for the Hardware-In-Loop simulations to evaluate the scheme is covered in Section V. Finally, Section VI presents a stability control case-study on the test-bed to demonstrate the and a dynamic thresholding scheme for the diagnosis process along with implementations of both invasive and opportunistic diagnostics.

II. Related Work

General Motor's OnStar, Ford's MyTouch, BMW Assist, Kia's Uvo [7] are examples of basic diagnostic services for remote vehicle monitoring and tracking. They are capable of alerting users of safety and performance issues, but the final diagnostics still has to be at the service center. While these systems provide early warnings for issues with the vehicle's operation, they are built upon the limited OBD DTCs for vehicle software and have no observability of the vehicle's control systems and control software.

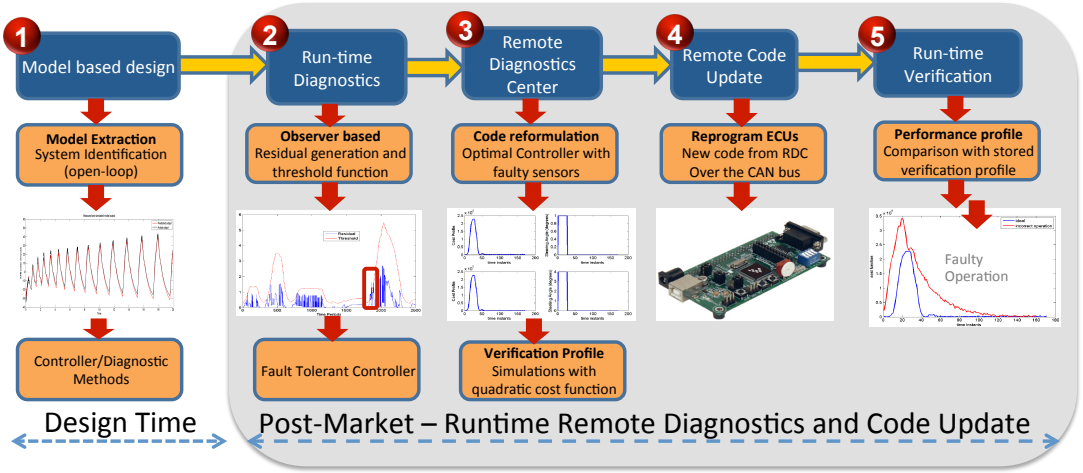

Fig. 2. Stages of the AutoPlug architecture

The use of event trace data which is logged at runtime and analyzed at the lab *aposteriori*, is a common approach called 'Record and Replay' debugging. AVEKSHA [8] improves upon this for tracing events in a real-time system and diagnosing possible timing related bugs in embedded software while the sensor node is deployed. In our previous work [9], we used logged data from the AutoPlug testbed to diagnose a PID Controller used as the Engine Idle-Speed Controller. Our current work extend this to analyze control systems between in-vehicle diagnostic data and the RDC.

On a different time scale, Pattipati et al. [10] use a data-driven approach over years of fault data to detect and determine faults in automotive systems. Classical Fault Detection and Tolerance methods in the context of aircraft control are reviewed by Huo et al. [11]. For adaptive controllers, ORTEGA [12] presents a state-space based approach to switch between controllers from high-performance to less performing but high-assurance controllers for fault tolerance in real-time control systems. Our approach relies on the fault diagnosis to decide which controller to switch to and also provides reconfiguration of the controller that is not based on a static set of faults.

NCSWT [13] is a software based tool for networked control systems which can be used for accurate simulation and evaluation of timing faults. While NCSWT is a general tool, the AutoPlug test-bed (Section V) used for evaluation in this paper is specifically for networked automotive control systems and also employs hardware in the loop.

Finally, firmware over the air (FOTA) updates have been introduced for infotainment systems in vehicles [14]. Firmware updates can be downloaded onto a smartphone which can also connect to the vehicles communication bus through the On-Board Diagnostics port (OBD). Similar methods for update of control software or for the code in any other safety critical ECU have yet to be developed and tested. Internet Connectivity for a vehicle has been used for personalized tuning of the vehicle in CarMA [15].

III. The AutoPlug Architecture

Regular firmware updates over a network for infotainment systems in vehicles is now a rising trend [14]. A central node keeps track of the firmware version for a large number of cars and when a new version is ready to be launched, updates the firmware code on each vehicle based on its difference with the existing version on the vehicle. Similar methods for the control systems and diagnostics code implemented on board the ECUs of vehicles have yet to be developed and implemented. This is because the safety-critical nature of the control systems implemented on the ECUs makes the task of coming up with these methods and proving their correctness more exacting.

The working of the AutoPlug scheme can be divided into five steps, as shown in Fig. 2. At the design stage, a model for the automotive system is extracted and is used for the controller formulation and the model-based diagnostic methods. After the vehicle has been deployed, run-time diagnostics onboard the vehicle detect and isolate faults, and the knowledge of that is used to switch to a fault tolerant controller. The diagnostic data logged in the vehicle is then sent to the RDC where the new controller/diagnostics code is formulated. The reformulated code, along with a verification profile for run-time evaluation is sent to the vehicle and reprogrammed onto the ECUs.

This section will explain the key parts of the AutoPlug architecture shown in Fig. 3 and provide an overview of their functions.

A. Vehicle Dynamics and Sensors/Actuators

Measurements, like yaw rate and roll angle, are available as messages on the CAN bus through the corresponding sensors during vehicle operation. Actuators (e.g. throttle, brakes) can also be commanded by corresponding CAN messages. In addition, an approximation of the vehicle dynamics (e.g. powertrain dynamics, lateral dynamics) is available at the design stage. This knowledge of the vehicle dynamics and the run-time sensor measurements from the vehicle and actuation are used for the control and diagnostics onboard the vehicle.

B. Bank of Controllers

Onboard the ECUs, multiple controllers are implemented to achieve the same performance criteria while using different subsets of available sensors. Normal operation involves the nominal controller using all the available (and needed) sensors for feedback control, and the sensor status is monitored by the diagnostics. Based on the status of

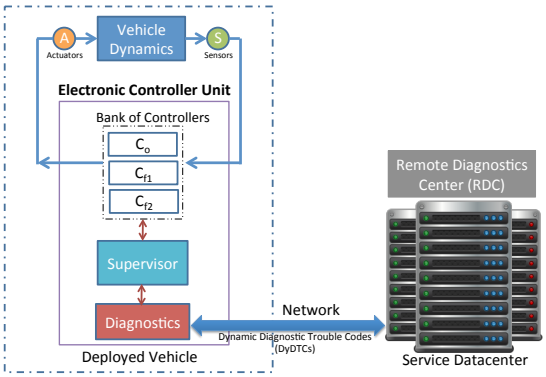

Fig. 3. The AutoPlug Architecture and its components

sensors, only one of the controllers is active at a time. This bank of controllers aims to achieve the predefined control objective despite a sensor or set of sensors providing faulty measurements. Each individual controller corresponds to one case of sensor fault(s) and is activated whenever that fault or set of faults is detected. The control objective may be met by these controllers through either *Accomondation* or *Reconfiguration* [16].

In the case that one or more sensor outputs are unreliable, one method of getting a fault tolerant feedback controller is to change the controller structure in order to ignore the outputs from the faulty sensors. This is called *Fault Accommodation*. Reconfiguration involves a change in the plant input-output structure while formulating the controller. AutoPlug architecture supports a bank of pre-formulated controllers which accommodate all combinations of sensor faults or a subset (for which fault accommodation leads to a stabilizing controller) of the combinations. Based on the information from the *Onboard Diagnostics*, the *Supervisor Logic* takes the original controller (for the no fault case) out of the loop and switches to the corresponding fault tolerant controller in run-time.

C. Onboard Diagnostics

Run-Time FDI is one of the principal components of the AutoPlug architecture. The onboard diagnostics code can be on a single ECU or distributed among different ECUs. The scheme which forms the backbone of the onboard diagnostics is tasked with monitoring sensor measurements and detecting any sensor faults and isolating the fault(s) to one or more sensors. A detailed explanation of the FDI scheme employed is outlined in section IV-B.

Because of the safety-critical nature of many feedback control systems in a vehicle and the related sensor data, detecting sensor faults is one of the foremost tasks of the onboard diagnostics system. Schemes for sensor FDI are numerous have been extensively studied in [17]. The bank of observers scheme [18] is a model based FDI scheme which is well suited to multi-output linear systems, but the Luenberger Observer is less than ideal for systems with noisy outputs, which most real world systems invariably are. Using the Kalman Filter, a standard residual generation scheme [17] and a thresholding scheme we present in section VI, we extend the FDI scheme. This scheme is applicable to a noisy system with modeling uncertainty and is robust in identifying and isolating multiple simultaneous faults.

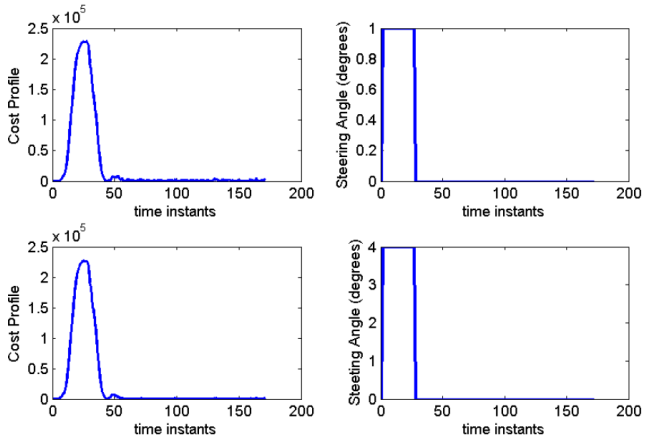

Fig. 4. Profile for verification of controller with lane change maneuvers of two different magnitudes of δ

In addition, we also modify the observer-based scheme for a real-time system where diagnostics may not always be scheduled periodically. This *Opportunistic Diagnostics* scheme is outlined in Section IV-D and is evaluated in section VI-F.

D. Supervisor

The supervisor in Fig. 3 is implemented onboard an ECU to overlook the functioning of the fault-tolerant architecture. The supervisor receives information from the onboard diagnostics module and switches between the original (no fault) controller and the fault-tolerant controllers based on which sensor or set of sensors is faulty. The supervisor is also responsible for logging run-time sensor measurements, control inputs, and diagnostics information for a finite window of time. This data logging continues for some time after the fault is detected, and is then transmitted to the Remote Diagnostics Center. In addition, after the remote controller code update, the supervisor is responsible for monitoring the performance of the new controller.

The run-time verification scheme is based on the fact that a controller, e.g. the stability controller of section VI is activated only for some particular maneuver, e.g. a lane change or a double lane change. Considering that the lane change is the maneuver we're interested in for verification purposes, a quadratic cost function (similar to that used for the initial optimal controller formulation) is used to generate a nominal operation profile from offline simulation. This offline profile is then compared to a running cost when the maneuver is being executed by the actual vehicle. It's not difficult to identify a lane change, and when the maneuver is started, the running cost is evaluated (after the maneuver is started) as

$$C(t) = \frac{x'_m(t)Qx_m(t)}{||x_m(t)||\delta} \tag{1}$$

Figure 4 shows the cost-profiles for maneuvers of two different magnitudes. Note that the profile generated from Equation 1 is normalized for the magnitude, and that is reflected in figure 4.

E. The Remote Diagnostics Center (RDC)

The RDC is a facility of the vehicle's manufacturer that is connected to all deployed vehicles and provides remote recalls management. The RDC performs tasks that may be autonomous or have a human in the loop. In the scope of this paper, the RDC performs autonomous tasks like controller reformulation and generation of a verification profile for patched controller evaluation onboard the vehicle at run-time. The RDC receives data from the vehicle whenever the vehicle encounters a fault. The logged data from the vehicle is used for the diagnostic tasks performed at the RDC and the reformulated controller is sent to the vehicle to be reprogrammed onto the ECUs.

After a fault has been detected in a sensor onboard the vehicle, logged data (before and after the fault) about the vehicle performance and the diagnostics (i.e. the observer-based state estimates and residuals) are sent to the RDC. At the RDC, the first and foremost task is to find out whether the fault is indeed a sensor fault. The other possibility is that wear-and-tear on the vehicle has led to changed parameters in the car model (e.g. suspension stiffness, cornering coefficients). This possibility is verified or ruled out by simulating the existing model with the logged inputs and comparing the outputs of this simulation to the logged outputs. If there is indeed a change in vehicle parameters, system identification is performed using the structure of the original model and the logged data to get a new plant model for that particular vehicle. This model is then used to generate new controller and diagnostics code for that vehicle. Irrespective of whether there is a sensor fault or a change in vehicle parameters, a new optimal controller is formulated and a performance profile for that controller is generated. This new control controller is coded and sent to the vehicle, along with the performance profile for run-time verification of the new controller.

IV. A Simple Example to Illustrate the Diagnostics Scheme

The model-based framework outlined in the previous section lends itself well to the case of linear systems. Consider a Linear Time Invaraint (LTI) discrete-system with n states, m outputs and p inputs.

$$x(k) = Ax(k-1) + Bu(k-1) + w(k-1), \tag{2a}$$

$$y(k) = Cx(k) + v(k) \tag{2b}$$

Here, $x \in \mathbb{R}^n$ is the state-vector, $u \in \mathbb{R}^p$ is the input-vector and $y \in \mathbb{R}^m$ is the output-vector of the system. The system has process noise $w \in \mathbb{R}^n$ and measurement noise $v \in \mathbb{R}^m$ which are assumed [19] zero-mean Gaussian and satisfy:

$$E[w(k)w'(k)] = Q; E[v(k)v'(k)] = R; E[w(k)v'(k)] = N \tag{3}$$

A. The Linear Bicycle model

For example, consider the lateral dynamics of a vehicle [20] represented by the continuous-time state space system given in Eq. 4. This model is called the 2-Degree of Freedom model or the Bicycle model and is popular in control literature for analyzing the lateral stability of a vehicle. The two degrees of freedom here are the vehicle lateral position y and the vehicle yaw angle ψ (which are also the states of the system). The vehicle is assumed to have a constant longitudinal velocity v_x. Eq. 4 shows the evolution of the rate of change of the lateral position and the yaw angle with braking force F_{BS} and steering angle δ being inputs to the system. As seen in Fig. 5, the vehicle lateral position is measured along the lateral axis of the vehicle to the center of rotation of the vehicle and the yaw angle is measured with respect to the global x-axis.

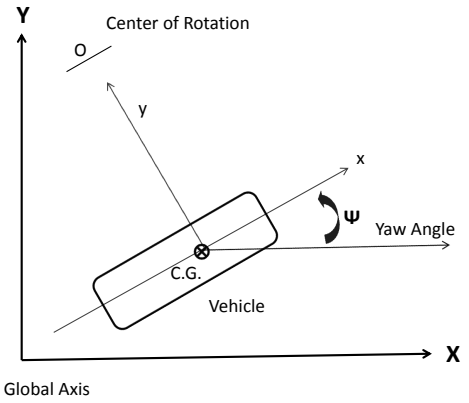

Fig. 5. The Two Degree of Freedom model

TABLE I
VEHICLE PARAMETERS

Vehicle mass	m	$1670kg$
Moment of Intertia	I_z	2100 kgm^2
Longitudinal Speed	v_x	27.78 m/s
C.G. to front wheel	a	0.99 m
C.G. to rear wheel	b	1.70 m
Cornering Coefficient(front)	C_{a1}	-123190 N/rad
Cornering Coefficient(rear)	C_{a2}	-104910 N/rad
Track width	T	1.52 m

$$\begin{bmatrix}\ddot{y}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\frac{C_{a1}+C_{a2}}{mv_x} & \frac{-bC_{a1}+aC_{a2}-mv_x^2}{mv_x}\\ \frac{aC_{a1}-bC_{a2}}{I_z v_x} & \frac{a^2C_{a1}+b^2C_{a2}}{I_z v_x}\end{bmatrix}\begin{bmatrix}\dot{y}\\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}\frac{-C_{a1}}{m} & 0\\ \frac{-aC_{a1}}{I_z} & \frac{T}{2I_z}\end{bmatrix}\begin{bmatrix}\delta\\ F_{BS}\end{bmatrix} \tag{4a}$$

$$\begin{bmatrix}\dot{y}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\dot{y}\\ \dot{\psi}\end{bmatrix} \tag{4b}$$

The vehicle parameters in Eq. 4 are enumerated in table I. Assuming that both $\dot{y}$ and $\dot{\psi}$ are measurable (full state feedback), the C matrix in Eq. 2 is identity and the D matrix is a zero matrix. Values for the process noise and measurement noise covariances are assumed to be $Q = I$, $R = 0.05I$ and N is a zero matrix for simplicity.

To be consistent with a real digital implementation, we discretize the continuous time system in Eq. 4 for a sampling time of 0.002s and represent it as a discrete time state space system in Eq. 5. Note that after discretization, the C matrix is still an identity matrix and the D matrix is still a zero matrix.

$$\begin{bmatrix}\dot{y}(k)\\ \dot{\psi}(k)\end{bmatrix} = \begin{bmatrix}4.015*10^{-5} & -0.0004001\\ 1.546*10^{-5} & -5.418*10^{-6}\end{bmatrix}\begin{bmatrix}\dot{y}(k-1)\\ \dot{\psi}(k-1)\end{bmatrix} + \begin{bmatrix}-0.02235 & -2.084*10^{-7}\\ 0.007383 & 3.681*10^{-8}\end{bmatrix}\begin{bmatrix}\delta(k-1)\\ F_{BS}(k-1)\end{bmatrix} \tag{5a}$$

For the lateral stability controller, similar to [20], we implement a Linear Quadratic Regulator (LQR) as the state feedback controller. The aim of the stability controller is to intervene with the vehicle steering in case of oversteer or understeer, but only by using differential braking and not by

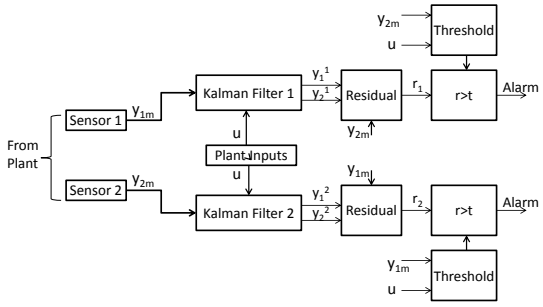

Fig. 6. The Fault Detection and Isolation Scheme. The redundant output estimates are used to find any discrepancy, in the form of a residual function, if a sensor is faulty. The threshold function acts as a decision function in the FDI scheme.

affecting the steering angle δ. The brake force input to the system is governed by the control law given:

$$F_{BS}(k) = -10^6 \begin{bmatrix} -1.0260 & -0.0562 \end{bmatrix} \begin{bmatrix} \dot{y}(k) \\ \dot{\psi}(k) \end{bmatrix} \quad (6)$$

B. The Diagnostics Scheme explained

For the applicability of the schemes outlined in Section III-C, we require the pair (A, C_i) to be observable $\forall i = 1, 2..n$, where C_i is the i^{th} row of the C matrix. This is because in the FDI scheme, there are n Kalman Filters, each driven by only one of the n-sensor outputs of the system and to estimate the states of the system from just the i^{th} sensor output requires (A, C_i) to be observable. The system given by the Eq. 5 meets these requirements, and shall be used throughout the section to illustrate the FDI scheme. The key idea behind the FDI scheme is to generate an analytical redundancy by having state estimates of the system from all the sensor outputs individually. This in turn means that there are now redundant output estimates (since $\hat{y} = C\hat{x}$) which when compared to the actual sensor measurements will help detect and isolate which sensor is faulty. Computationally, individual comparison of estimated signals is unwieldy and needs too many logical decision. To overcome this for the implementation of the scheme, a *residual* signal is generated as a function of the measurements and estimates. A fault is detected and isolated to a particular sensor if the residual crosses some *threshold*. Fig. 6 shows the overview of the scheme and Section IV-C explains the terms, notations and the working of the components of the figure.

C. The Kalman filter for residual generation

For a discrete time system given in Eq. 2, the Kalman filter is the optimal estimator. Given the plant and noise model in Eq. 2 and Eq. 3, the aim of the Kalman filter is to construct a state-estimator which minimizes the steady state error covariance [19]. The *state update* and *measurement update* steps of the Kalman filter can be represented in the form of a discrete state-space system:

$$z(k) = A_o z(k-1) + B_o \begin{bmatrix} y(k) \\ u(k) \end{bmatrix} \quad (7a)$$

$$\hat{x}(k) = C_o z(k) + D_o \begin{bmatrix} y(k) \\ u(k) \end{bmatrix} \quad (7b)$$

Here, z is the state of the Kalman filter, $\hat{x}(k)$ is the state estimate from the Kalman filter driven by system output y at time instant k given all measurements (output y and input u) upto time instant k. The reader can refer to [19] for the construction of matrices A_o, B_o, C_o and D_o.

For the 2-DOF model, the State and Output equations corresponding to Eq. 7 for the Kalman filter driven by the first output (lateral velocity $\dot{y}$, represented as y_1) of the system in Eq. 5 are:

$$z(k+1) = \begin{bmatrix} -0.0001034 & -0.0004001 \\ 1.087 * 10^{-6} & -5.418 * 10^{-6} \end{bmatrix} z(k) + \begin{bmatrix} -0.02235 & -2.084 * 10^{-7} & 0.0001436 \\ 0.007383 & 3.681 * 10^{-8} & 1.437 * 10^{-5} \end{bmatrix} \begin{bmatrix} \delta(k) \\ F_{BS}(k) \\ y_1(k) \end{bmatrix} \quad (8a)$$

$$\hat{x}^1(k|k) = \begin{bmatrix} 0.1668 & 0 \\ 0.2752 & 1 \end{bmatrix} z(k) + \begin{bmatrix} 0 & 0 & 0.8332 \\ 0 & 0 & -0.2752 \end{bmatrix} \begin{bmatrix} \delta(k) \\ F_{BS}(k) \\ y_1(k) \end{bmatrix} \quad (8b)$$

Here, $\hat{x}^1(k|k)$ is the state estimate from the Kalman filter driven by system output y_1 at time k given all measurements (output y_1 and input u) upto time instant k. Output estimates $y^1(k|k)$ can be calculated by simply multiplying the C matrix of the system in Eq. 5, which is identity, with the state estimate $\hat{x}^1(k|k)$. With a similar formulation for the Kalman filter driven by system output y_2, output estimates $y_1^2(k|k)$ and $y_2^2(k|k)$ are obtained. To reduce notational clutter, estimates $y_i^j(k|k), \forall i, j$ are simplified as $y_i^j(k)$.

If there's no fault in the system, the output vector estimate from each Kalman filter should be in close agreement with the measurements. In case there's a fault in one of the sensors, the output estimate from that sensor will differ from the measured sensor values. A residual function [17] to represent this analytical redundancy for the Kalman filter driven by system output j is:

$$r_j(k) = \prod_l |(y_l(k) - \hat{y}_l^j(k))|, \forall l = 1, 2, ...m, l \neq j \quad (9)$$

Fig. 7 shows the sensor measurements and the corresponding residuals for normal operation. The jitter is modeled as a time varying delay, which is Gaussian distributed with a mean of three sampling periods and a variance of four sampling periods. Introducing the jitter in sensor 1 (lateral velocity) measurements, it's easy to notice in Fig. 8 that the residual for sensor 1 is now around three times higher than it was for the no fault case. The flip-side to using this residual scheme is that now the residual for sensor 2 (yaw rate) is

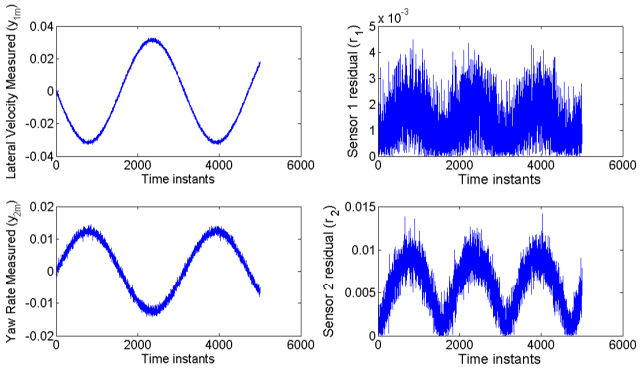

Fig. 7. No-fault Case: Measurements (on the left) and residuals (on the right) for the case where no sensor is faulty. The residual is ideally small in this case.

also higher (by around three times) than it was for the no fault case, even though the fault is just in sensor 1.

The increase in residuals implies that fault detection with this standard residual scheme is easy, but isolating the fault to a particular sensor is not. This is because the residual for the non-faulty sensor is also increasing. To remedy this, we introduce a scheme in Section VI to generate thresholds which are robust to faults in other sensors and make isolation of faults easier.

D. Opportunistic Diagnostics

In a real-time operating system (for example, we use nano-RK [21]) running on an ECU, control functions and diagnostics are implemented as *tasks*. These tasks have a *period*, *execution time* and a *deadline*. In the case of a periodically sampled controller or observer, the period and deadline is equal to the sampling period, while the execution time has to be less than the sampling period in order for the task to meet its deadline. Consider the brakes ECU of the AutoPlug testbed in Section V. The ECU may have multiple control tasks like the stability controller, ABS and traction control and its utilization may be such that periodically scheduling the observer based diagnostics may lead to some control task missing its deadline. In this case, periodic scheduling of diagnostics is infeasible since we do not want the diagnostics to interfere with the safety critical control systems.

The standard Kalman filter based residual scheme of Section IV-C, when implemented on an embedded controller, corresponds to the case of periodic execution of the observer as a task. The LQR stability controller (Section IV) is also a periodic task. On the AutoPlug testbed (Section V), the LQR controller execution time was found to be 0.260ms, while that for the Kalman filter was 0.740ms on the brakes ECU, which is a HCS12 microcontroller. Since the testbed has a sampling period of 2ms, the controller should finish all its computations within that period in order to not miss its deadline and the next measurement sample. With the Kalman filter's significantly higher execution time and traction control and ABS adding a computational overhead, it makes more sense for the Kalman filter based diagnostics to be executed only when there is *slack* available [22] in the ECU. This leads to an aperiodically executed Kalman filter and we call this the *Opportunistic Diagnostics* scheme.

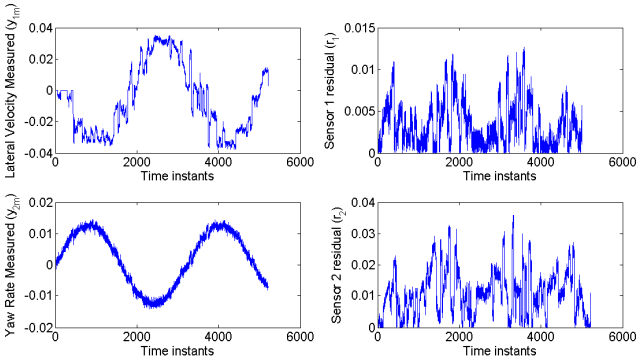

Fig. 8. Faulty Sensor Case: Measurements (on the left) and residuals (on the right) for jitter in sensor 1. The increased residuals indicate that there is a fault, but due to the similar increase in residuals for both sensors, isolation is difficult

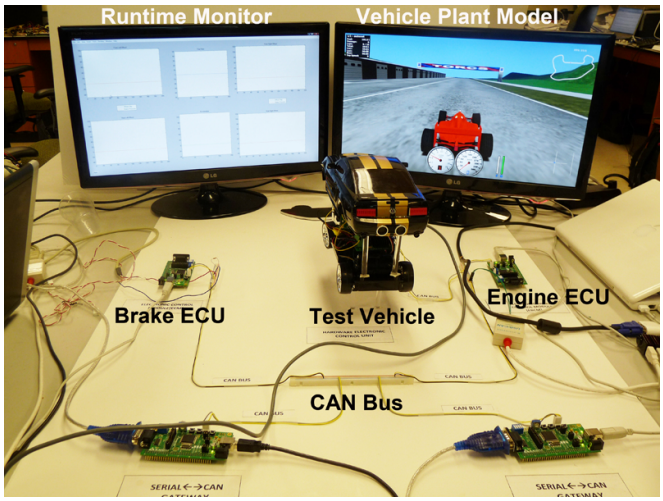

Fig. 9. The AutoPlug testbed

Since storing measurement samples at every period doesn't involve much computational overhead, we can use the stored data and simply modify the Kalman filter to account for the periods of non-execution as shown in Eq. 10. This modification essentially makes the Kalman filter aperiodic in execution, but with periodically sampled data. An alternative to storing all measurements is dropping them in the periods that the Kalman filter is not executed in [23]. In this paper we, explore the simpler approach where all measurements are stored.

$$z(k+p) = A_o^p z(k) + \sum_{j=0}^{p-1} A_o^{p-1-j} B_o \begin{bmatrix} y(n+j+1) \\ u(n+j+1) \end{bmatrix} \tag{10a}$$

$$\hat{x}(k+p) = C_o z(k+p) + D_o \begin{bmatrix} y(k+p) \\ u(k+p) \end{bmatrix} \tag{10b}$$

Here, k is the time instant in which the Kalman filter was previously executed, and $k+p$ time periods is the time instant at which the Kalman filter is executed next. A buffer stores the measured outputs y and inputs u from time k to time $k+p$. The generation of residuals is such that the residual is zero when there is no execution and given by Eq. 9 when the diagnostics task is executed on the embedded controller. This opportunistic diagnostics scheme is evaluated with Rate Monotonic Scheduling in Nano-RK and the results are presented in Section VI.

V. The AutoPlug testbed

To test out the framework outlined in the previous sections on a system with real hardware, we implement the FDI scheme on the AutoPlug testbed (see Fig. 9) [1]. The AutoPlug testbed is a Hardware-In-The-Loop simulation platform for ECU development and testing. The hardware is in the form of a network of ECUs, on which we implement the control and diagnostic algorithms. Each ECU runs the nano-RK RTOS [21], a resource kernel with preemptive priority-based real-time scheduling. Instead of a real-vehicle, our

[1]The test-bed was demonstrated at CPSWeek 2011

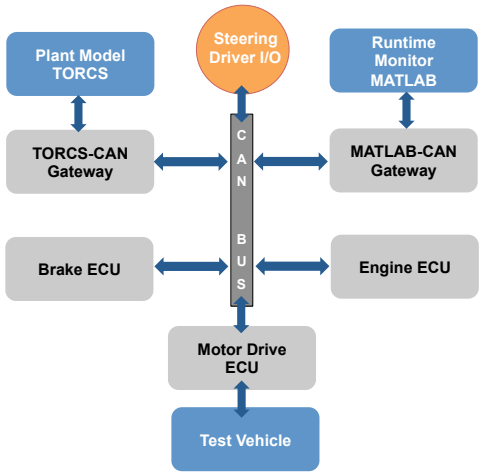

Fig. 10. AutoPlug System Architecture

plant uses The Open-source Race Car Simulator (TORCS). This provides physics-based high-fidelity vehicle models and different road terrains. The testbed provides us with the realism of using a real vehicle, and also has enough flexibility to implement our own code. In addition, we can introduce faults which are not covered by set of standard Diagnostic Trouble Codes (DTC). We have tested out basic control algorithms, running as real-time tasks on nano-RK, for Anti-Lock Braking System (ABS), Traction Control, Cruise Control and Stability Control to see that the testbed indeed performs like a real vehicle would. AutoPlug is free and open-sourced [24].

A. Testbed architecture

The AutoPlug testbed consists of three layers, *Vehicle Dynamics Simulation*, *ECU Network* and the *middleware* for control algorithms, runtime software/system diagnosis, upgrade and verification. The simulation layer models the dynamics of a vehicle (e.g. Toyota Corolla) on the physics-based simulator (TORCS). The ECU network consists of four embedded controllers (FreeScale HCS 12) networked over an industry standard CAN bus. The middleware is a small computer that provides a gateway protocol for vehicle manufacturers to interface with the ECU network and provides us with the simulated capabilities of a RDC. Fig. 10 shows a simplified view of the AutoPlug architecture. The four ECUs on the testbed are:

(1) TORCS Gateway ECU: The simulation data (sensor values) are sent from TORCS (running on a computer) to the ECUs in real time over the CAN bus via this gateway. The inputs to the simulation in TORCS, from the ECUs and the user, are also received through the TORCS gateway and sent to the simulator.

(2) Brakes ECU: This ECU controls the inputs to the brakes of the vehicle in TORCS. Based on the user inputs (which are sent over the CAN bus) and sensor values, the control algorithm on the ECU calculates the brake force for the individual wheels of the vehicle (generalized as the left or the right side of the vehicle). This is sent as a message over the CAN bus to the TORCS Gateway which in turn sends the calculated inputs to the vehicle actuator in TORCS.

(3) Engine ECU: This ECU controls the acceleration, gear and clutch inputs to the vehicle. Algorithms like Cruise Control and Traction control are implemented onboard this ECU and it's tasked with modulating the user inputs to the vehicle and sending the modulated inputs as a message to the TORCS Gateway over the CAN bus.

(2) MATLAB Gateway: In order to log data from the testbed and to monitor the vehicle parameters (like yaw rate etc.) in real-time , the MATLAB gateway reads the TORCS outputs, user inputs, and controlled inputs from the Brakes and Engine ECU from the CAN bus and sends them to the middleware. The middleware can also communicate with the other ECUs over the CAN bus if needed.

B. Code updates

Reprogramming the ECUs over the CAN bus was one of the features of the first generation AutoPlug framework [24]. A smartphone can connect to the vehicle's CAN bus through the Onboard Diagnostics (OBD) port and the code update can be downloaded onto the smartphone. This downloaded code can then programmed onto the ECU over the CAN bus. Once a fault is detected onboard the vehicle, the Remote Diagnostics Center receives information of that fault, and new code is formulated at the RDC. The vehicle user then has a choice to initiate a code update remotely. The downloaded code is loaded in a secure manner from the owners smartphone into the vehicle via a WiFi gateway interfacing the vehicles on-board diagnostics (OBD) computer. This performance of this code is then evaluated at runtime (Section III-E).

VI. Case Study: Vehicle Lateral Dynamics on the AutoPlug Testbed

Electronic Stability Control (ESP) has been a safety feature on vehicles for a number of years [25]. The stability controller is tasked with maintaining the vehicle at the desired yaw angle corresponding to the driver's steering input. A common way of doing this by using differential braking which introduces a yaw rate corresponding to the brake force applied to either the inner or outer wheels. This yaw rate is used to steer the vehicle onto the desired track. Pilutti et al. [20] compare the performance of three classical control techniques (PID, LQ and pole-placement) applied as stability controllers on a 2-DOF Bicycle vehicle model.

A. 3-DOF Linear model for vehicle lateral dynamics

TORCS, is a physics based vehicle dynamics simulator. It is difficult to accurately represent the vehicle dynamics modeled in TORCS with the simple Bicycle model. On the other hand, higher order models [26] have been studied for vehicle dynamics control, but their complexity makes them a poor match for the Off-The-Shelf embedded controllers used in a vehicle. D'Silva et al. [27] add another degree of freedom, Roll, to the Bicycle model and introduce a 3-DOF linear model for vehicle lateral dynamics. We add a slight modification to 3-DOF model to include the effect of brake forces to the left and right sides of the vehicle. The discrete-time version of this model relating the lateral velocity, roll angle and yaw rate dynamics is used as a reference for the remainder of the text.

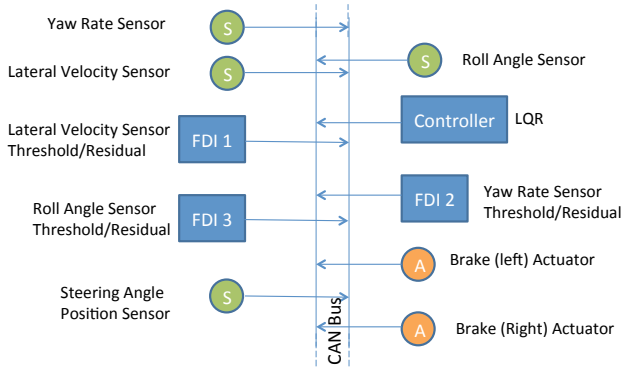

Fig. 11. Networked Control System View of the testbed

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} \beta_1 & \beta_2 & 0 & \beta_3 \\ \beta_4 & \beta_5 & 0 & \beta_6 \\ \beta_7 & \beta_8 & \beta_9 & \beta_{10} \\ 0 & 0 & \beta_{11} & \beta_{12} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} \beta_{13} & 0 & 0 \\ \beta_{14} & \beta_{15} & \beta_{16} \\ \beta_{17} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta(k) \\ F_l(k) \\ F_r(k) \end{bmatrix} \tag{11a}$$

$$\begin{bmatrix} \dot{y}(k) \\ \dot{\psi}(k) \\ \dot{\phi}(k) \\ \phi(k) \end{bmatrix} = \begin{bmatrix} \beta_{18} & 0 & 0 & 0 \\ 0 & \beta_{19} & 0 & 0 \\ 0 & 0 & \beta_{20} & 0 \\ 0 & 0 & 0 & \beta_{21} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} \tag{11b}$$

Eq. 11 shows the structure of discrete-time 3-DOF linear model for vehicle lateral dynamics. $\beta_i, \forall i = 1, 2, ..., 21$ are the free parameters of this model which are dependent on the characteristics (like mass, length etc.) of a particular vehicle, the other parameters in the state-space model are 0. These free parameters relate the dependence of the evolution of the states of the system upon each other and the inputs to the system. Here, x_1, x_2, x_3, x_4 are the 4 states of the vehicle, scaled versions of which (by $\beta_{18}, \beta_{19}, \beta_{20}, \beta_{21}$) are the lateral velocity $\dot{y}$, yaw rate $\dot{\psi}$, roll rate $\dot{\phi}$ and the roll angle ϕ.

B. System Identification: 3-DOF Linear model from TORCS

TORCS is a physics based vehicle dynamics simulator and is used in the AutoPlug testbed to provide the vehicle dynamics. In order to apply conventional control and observer techniques on the testbed, a linear model for the vehicle dynamics in TORCS is required. Since we're interested in stability control, we use the 3-DOF model for lateral dynamics in Eq. 11 as a starting point and perform a structured system identification to estimate the free parameters (β_i) for the vehicle dynamics in TORCS. Fig. 12 shows the measured outputs from the testbed and those from the identified model for the same set of inputs (also in the figure). Eq. 12 shows the identified model (for a sampling time of 0.002s). Note that the outputs from TORCS are only the lateral velocity $\dot{y}$, yaw rate $\dot{\psi}$ and the roll angle ϕ. To obtain the remaining output, the roll rate $\dot{\phi}[k]$, we take the difference of $\phi[k]$ and $\phi[k-1]$. Fig. 11 shows a Networked Embedded Control Systems (NECS) view of the setup for the case study.

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} 0.98893 & 0.0074715 & 0 & -0.043095 \\ -0.20552 & 0.95266 & 0 & -0.089656 \\ 0.0047131 & -0.00083028 & 0.71306 & -0.071009 \\ 0 & 0 & -0.56114 & 0.8176 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} -0.014049 & 0 & 0 \\ 0.068138 & 2.547e-7 & -2.5052e-7 \\ 0.00088488 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ F_l \\ F_r \end{bmatrix} \tag{12a}$$

$$\begin{bmatrix} \dot{y}[k] \\ \dot{\psi}[k] \\ \dot{\phi}[k] \\ \phi[k] \end{bmatrix} = \begin{bmatrix} -0.10859 & 0 & 0 & 0 \\ 0 & 1.231 & 0 & 0 \\ 0 & 0 & -0.041089 & 0 \\ 0 & 0 & 0 & 0.5834 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} \tag{12b}$$

Similar to the approach in section IV, the stability controller for the 3-DOF model is an infinite-horizon LQR. The inputs to the plant from the controller follow the state-feedback scheme:

$$\begin{bmatrix} F_l(k) \\ F_r(k) \end{bmatrix} = -\begin{bmatrix} -1.5335 & 2.3662 & 7.2193 & -3.7494 \\ 1.5081 & -2.3269 & -7.0995 & 3.6872 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} \tag{13}$$

Note here, that the states can be obtained from the measurements simply by dividing the measurements by the diagonal elements of the C matrix of the identified model in Eq. 12.

$$x = \begin{bmatrix} \frac{\dot{y}(k)}{-0.10859} & \frac{\dot{\psi}(k)}{1.231} & \frac{\dot{\phi}}{0.041089(k)} & \frac{\phi(k)}{0.5834} \end{bmatrix}' \tag{14}$$

To simplify the notations, the state vector is $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}'$, input vector is $u = \begin{bmatrix} \delta & F_l & F_r \end{bmatrix}'$ and the output vector is $y = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}'$ where y_1, y_2, y_3 and y_4 are $\dot{y}$, $\dot{\psi}$, $\dot{\phi}$ and ϕ respectively.

C. Detecting and Isolating sensor faults

The scheme for detecting sensor faults for the ESP implemented on the AutoPlug testbed follows the same steps as in section IV. The Kalman filter driven by the lateral velocity sensor is:

$$\hat{x}^1(k+1|k) = \begin{bmatrix} 0.9871 & 0.007472 & 0 & -0.0431 \\ -0.1929 & 0.9527 & 0 & -0.08966 \\ 0.004697 & -0.0008303 & 0.7131 & -0.07101 \\ -2.747e-005 & 0 & -0.5611 & 0.8176 \end{bmatrix} \hat{x}^1(k|k-1) + \begin{bmatrix} -0.01405 & 0 & 0 & -0.01687 \\ 0.06814 & 2.547e-007 & -2.505e-007 & 0.1158 \\ 0.0008849 & 0 & 0 & -0.0001495 \\ 0 & 0 & 0 & -0.0002529 \end{bmatrix} \begin{bmatrix} u(k) \\ y_1 \end{bmatrix} \tag{15a}$$

$$\begin{bmatrix} \hat{y}_1^1(k|k) \\ \hat{x}^1(k|k) \end{bmatrix} = \begin{bmatrix} -0.1084 & 0 & 0 & 0 \\ 0.998 & 0 & 0 & 0 \\ 0.01278 & 1 & 0 & 0 \\ 1.781e-006 & 0 & 1 & 0 \\ -3.237e-005 & 0 & 0 & 1 \end{bmatrix} \hat{x}^1(k|k-1) + \begin{bmatrix} 0 & 0 & 0 & 0.00195 \\ 0 & 0 & 0 & -0.01796 \\ 0 & 0 & 0 & 0.1177 \\ 0 & 0 & 0 & 1.64e-005 \\ 0 & 0 & 0 & -0.0002981 \end{bmatrix} \begin{bmatrix} u(k) \\ y_1(k) \end{bmatrix} \tag{15b}$$

The Kalman Filters for the two other measured outputs (yaw rate and roll angle) can be formulated as in section IV-C. The output estimates needed for generating the residuals as in Eq. 9 can be obtained by multiplying the state estimates by the C matrix of Eq. 12. The residual functions for the three sensors are given in Eq. 16, where the notations have the same meaning as in section IV-C.

$$r_1 = |(y_2 - y_2^1)(y_4 - y_4^1)| \tag{16a}$$

$$r_2 = |(y_1 - y_1^2)(y_4 - y_4^2)| \tag{16b}$$

$$r_4 = |(y_1 - y_1^4)(y_2 - y_2^4)| \tag{16c}$$

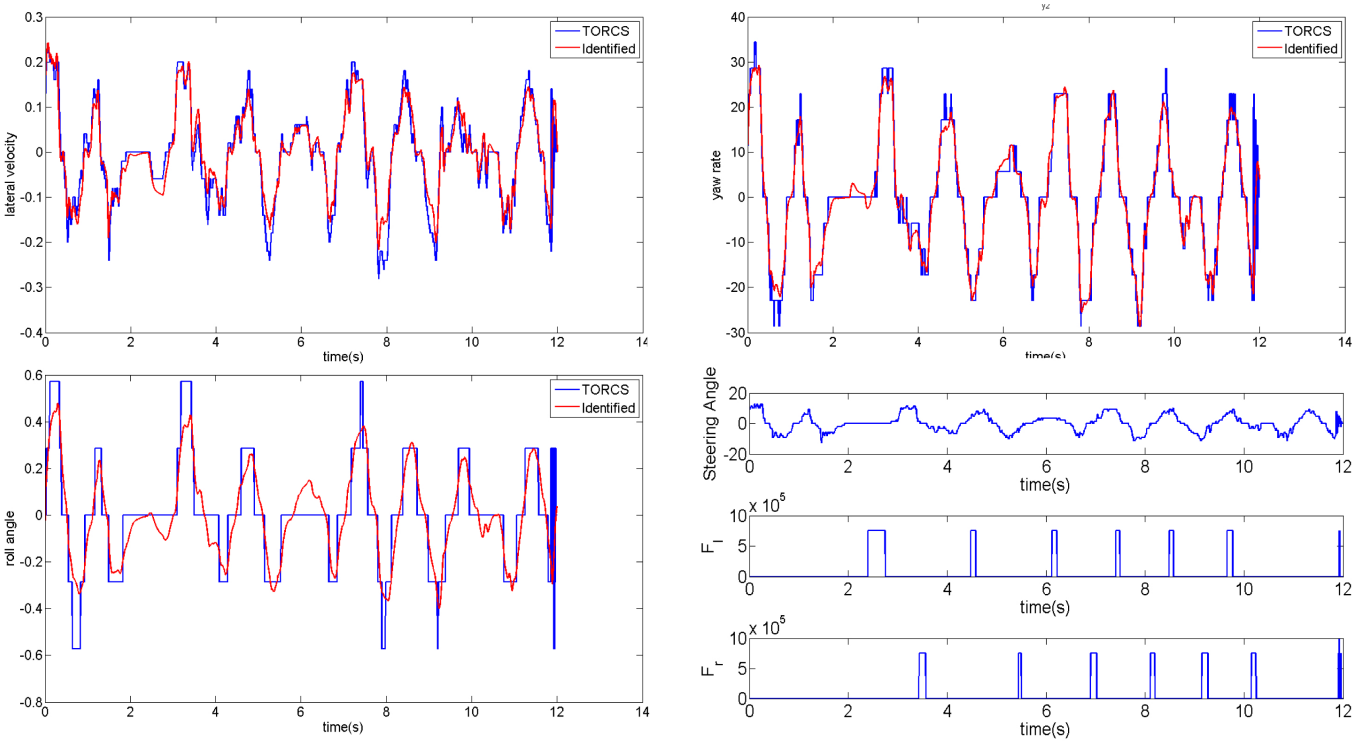

Fig. 12. Outputs from TORCS vs outputs from identified model for the same inputs. These results from the open loop system identification show a good match between TORCS and the identified model.

D. Dynamic Thresholding of residuals

As is evident from Eq. 16 and Fig. 8, the residual for a sensor increases in value even if there is a fault in one of the other sensors. It's also seen [17] that the modal content of the residuals resembles that of the inputs. This property is also seen from Fig. 7 where the steering angle input is a sinusoid. In [17] this analysis was done for a system with plant disturbances, or imperfect knowledge of the system model. For the current case study, this holds as the identified model is not perfect. The simple but innovative thresholding scheme outlined in [17] is difficult to apply to a system with measurement noise, in addition to plant disturbances, as is generaly the case in a real world system, and also in our setup. The thresholding function outlined in this section is developed in order to allow for detection and isolation of the fault to a particular sensor. Another advantage with the scheme outlined here, with respect to a Networked Architecture (Fig. 11) is that the decision of isolating a fault to a particular sensor does not need shared knowledge of other residual functions or observer estimates [27], and is also applicable to the case of multiple simulataneous faults while. Also, the scheme is computationally simple which allows it to be implemented on off-the-shelif embedded systems. For a system with m outputs and p inputs, the proposed threshold function for the i^{th} sensor is:

$$T_i(k) = |\mathcal{F}(\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathcal{T}_i, \mathcal{U}_l, \mathcal{Y}_j, C, N)| \tag{17a}$$

$$\mathcal{U}_l = (u_l(k), u_l(k-1), ..., u_l(k-N)), \forall l = 1, ..., p, l \neq i \tag{17b}$$

$$\mathcal{Y}_j = (y_j(k), y_j(k-1), ..., y_j(k-N)), \forall j = 1, ..., m, j \neq i \tag{17c}$$

Here, $\boldsymbol{\alpha} \in \mathbb{R}^{N-1}$, $\boldsymbol{\beta} \in \mathbb{R}^N$ and $C \in \mathbb{R}$. Fig. 8 suggests that the residual of the non-faulty sensor follows a pattern similar to the measurement from the faulty sensor. Eq. 9 shows that the residual is indeed dependent on the measurements from other sensors. Combining this with the fact that the residual also shares similarities with the uncontrolled input (the steering angle) of the example in Section IV, the threshold is made a function of both the input and the measurements from other sensors. Based on the results seen in the bicycle model example, and also from the AutoPlug testbed, the threshold was also made a function of some previous measurements $\mathcal{Y}_j$ and inputs $\mathcal{U}_l$, and also of some previous threshold values $\mathcal{T}_i$. This intuition leads to the general formulation for the threshold function in Eq. 17.

The function F that we chose in this case study is an Autoregressive model of order N, and the threshold function is of the form:

$$T_i(k) = |\sum_{l=0}^{l=N} \beta_l [u(k-l) + \prod_{j=1, j\neq i}^{j=m} y_j(k-l)] - [\sum_{l=1}^{l=N} \alpha_l T_i(k-l)] + C| \tag{18}$$

Appropriately choosing C, N, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ results in a decision function where a fault in sensor i is reported if $r_i \geq T_i$ for some k periods of time or occurs with a high frequency in some time window. Here, k is another decision variable to be chosen, one way to do which can be to have a reasonably large value of k if the system is sampled fast or a small value if the sampling rate is low. Note, the threshold function is meant to detect and isolate faults which manifest themselves for more than just a few instants, and aren't just outliers. So far the choice of C, N, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ has been done experimentally and has yielded reasonably good results. The same parameters, once chosen, for one sensor have worked well across different types of faults (stuck at, noise, calibration, delay, jitter), and across a wide range of input and measurement values. Some specific results of this are provided in Section VI-E.

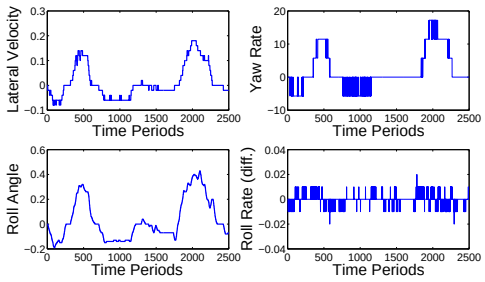

Fig. 13. System outputs (closed loop)

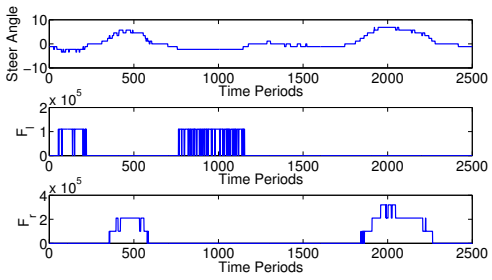

Fig. 14. Inputs to the system. Note, the brake forces are regulated by a feedback controller, while the steering input is from the vehicle driver.

E. Results from the Testbed

The no fault system measurements and control inputs are shown in Fig. 13 and Fig. 14 respectively, note here that the roll rate is not measured, but calculated by simply the difference of the roll angle at time period k and $k-1$. The Kalman Filter estimates (from the lateral velocity sensor) for the three measured outputs are shown in Fig. 15. Note, due to limitations of space, only results from the diagnostics of the the lateral velocity sensor are presented here.

1) Avoiding False alarms: To test the robustness of the threshold function to faults in another sensor, we introduce a calibration fault in the roll angle sensor. Fig. 16 shows the residual and threshold in this case for the Lateral Velocity Sensor, which is not faulty. The residual increases due to fault in the roll angle sensor, and as expected, the threshold also increases to accomodate that and not incorrectly isolate the fault to the Lateral Velocity Sensor. The false alarms when the residual does cross the threshold can be ignored by correctly choosing an appropriate value for the time window in which the fault should persist.

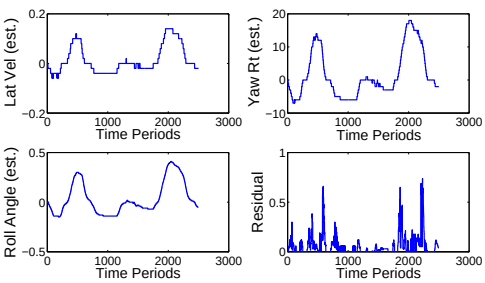

Fig. 15. Estimates from the Kalman Filter driven by sensor 1

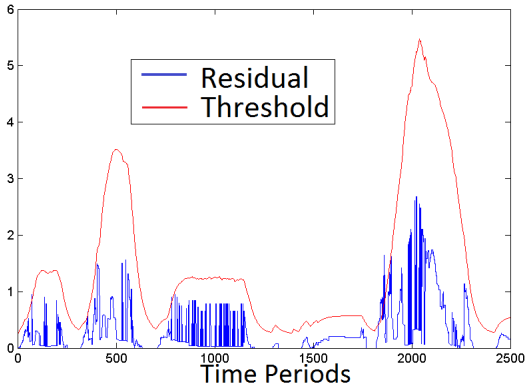

Fig. 16. Residual and threshold for lateral velocity sensor, with fault in roll angle sensor. Note how the threshold for sensor 1 increases to adapt to the increase in the residual due to fault in sensor 4

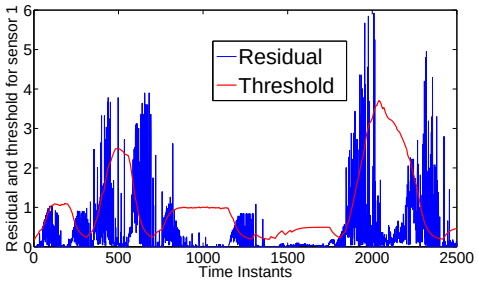

Fig. 17. Residual and threshold for jittery Lateral Velocity sensor

2) Real-Time faults: We introduce jitter in sensor 1, with the time varying delay (Gaussian distributed) having a mean of $0.004s$, variance of $.02s$ and being bound between $0s$ and $0.2s$. These are reasonable delays, considering that the sampling rate of the controller is $0.002s$. Fig. 17 shows the residual and threshold for sensor 1 in this case, where it is evident that the fault is indeed in sensor 1.

3) System faults: To experimentally test out the scheme, we introduced gain and offset (calibration) faults in the sensors needed for the stability control of the vehicle. Introducing a small gain and bias fault in the lateral velocity sensor, as expected, blows up the residual (but not the threshold), which results in the fault being detected (Fig. 18).

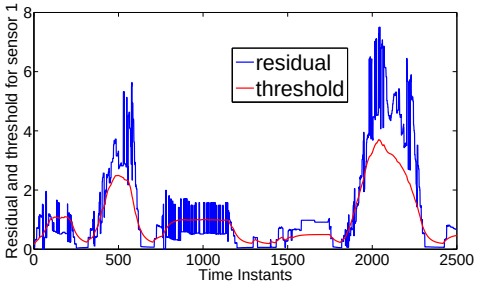

Fig. 18. Residual and threshold for lateral velocity sensor (calibration fault)

F. Opportunistic Diagnostics

The opportunistic diagnostics scheme was evaluated with the Nano-RK RTOS running on the ECUs. Of particular interest to us is the brakes ECU, which is responsible for the LQR stability controller, the ABS and traction control and also the diagnostics for the lateral velocity sensor. The control tasks are Rate Monotonic scheduled, which implies a fixed priority for all tasks, with the stability controller having the highest priority. The execution of the diagnostics task is made dependent on the slack available. Fig. 19 shows the time periods elapsed between two successive instants of the Kalman Filter based diagnostics being executed. Fig. 20 shows the residual for the lateral velocity sensor with opportunistic diagnostics, and also with the periodic diagnostics. It is seen that the same threshold function works with the opportunistic diagnostics as well.

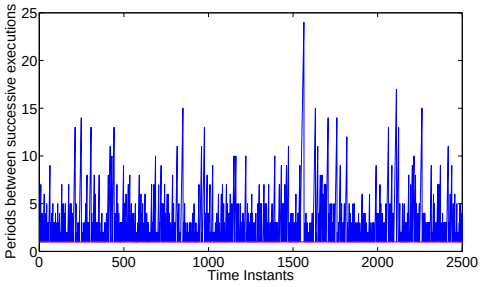

Fig. 19. Time periods elapsed between successive executions of the opportunistic diagnostics. The red line at y=1 shows the case of periodic execution.

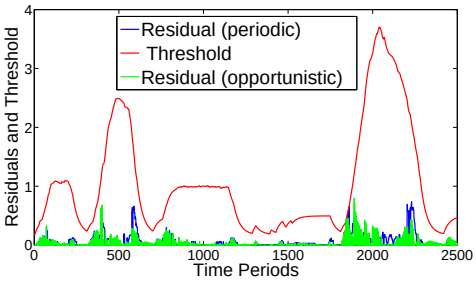

Fig. 20. Residuals and threshold for lateral velocity sensor (no fault)

Introducing jitter (with the same characterstics as in Section VI-E.2) in the lateral velocity sensor, Fig. 21 shows the residual and threshold with the opportunistic diagnostics. It can be seen that the fault is detected and isolated to the lateral velocity sensor.

VII. Conclusion

In the paper, we introduce a framework for merging onboard and remote diagnostics for Automotive Control Systems which can potentially make vehicle recalls a less reactive process. We also show a diagnostics scheme for a feedback control system and evaluate it on the AutoPlug testbed with reasonably good results. The overall scheme is relatively new, and potentially risky, but it can initially target non-critical control system, e.g. the vehicle body control systems. One particular example which shows that

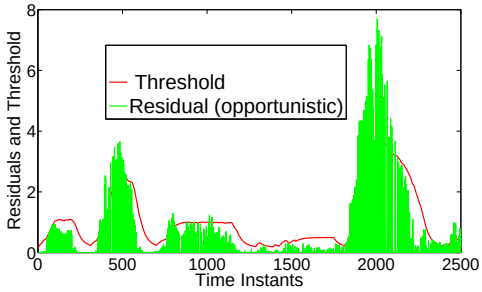

Fig. 21. Residuals and threshold for lateral velocity sensor (with jitter in sensor)

this scheme could be useful is the 936,000 vehicles that Honda had to recall due to issues with the power window and the automatic transmission [28]. Other possible issues may arise with the safety of the Firware-Over-The-Air (FOTA) approach for safety critical ECUs, where parties with malicious intent may hack into the network and compromise the safety of the vehicle by reprogramming the ECUs. So far, this has not been a part of our study, but Koscher et al. [29] have extensively studied the security of modern automobiles. Also, security of Cyber Physical Systems (CPS) [30], [31] is of growing interest. A logical extension of our work is to focus on CPS security for networked automotive control systems, a classification of possible attacks to look at is presented in [32]. A more specific extension is to formulate a method to tune the design parameters of the threshold function in diagnostics scheme, which currently are experimentally chosen.

References

[1] NHTSA Campaign ID number:09V218000. HTTP://WWW.SAFERCAR.GOV.

[2] Jaguar Software Issue May Cause Cruise Control to Stay On. http://spectrum.ieee.org/riskfactor/green-tech/advanced-cars/jaguar-software-issue-may-cause-cruise-control-to-stay-on.

[3] Honda recalls 2.5 million vehicles on software issue. http://www.reuters.com/article/2011/08/05/us-honda-recall-idUSTRE77432120110805.

[4] AUTOSAR Homepage. http://www.autosar.org/.

[5] J. Schaufalle and T. Zurawka. *Automotive Software Engineering*. SAE International, 2005.

[6] On-board Diagnostic Codes. http://www.obd-codes.com.

[7] Hephaestus Books. *Articles on Vehicle Telematics*. 2011.

[8] M. Tancreti and M. S. Hossain and S. Bagchi and V. Raghunathan. AVEKSHA: A Hardware-Software Approach for Non-intrusive Tracing and Profiling of Wireless Embedded Systems. *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011.

[9] U. Drolia and Z. Wang and Y. Pant and R. Mangharam. AutoPlug: An Automotive Test-bed for Electronic Controller Unit Testing and Verification. *Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems*, 2011.

[10] K. Pattipati, C. Sankavaram, B. Wang, P. Zhang, Y. Zhang, M. Howell, and M. Salman. Fault Diagnosis and Prognosis in a Network of Embedded Systems in Automotive Vehicles. *Position Paper for NSF-NIST-USCAR Workshop on Cyber-Physical Systems*, 2011.

[11] P. Ioannou Y. Huo and M. Mirmirani. Fault-Tolerant Control and Reconfiguration for High Performance Aircraft: Review. *CATT Technical Report*, 2001.

[12] X. Liu, K. Lee, Q. Wang, and L. Sha. ORTEGA: An Efficient and Flexible Software Fault Tolerance Architecture for Real-Time Control Systems. *IEEE Transactions on Industrial Informatics*, 2008.

[13] E. Eyesi and J. Bai and D. Riley and J. Weng and Y. Xue and X. Koutsoukos and J. Sztipanovits. NCSWT: an integrated modeling and simulation tool for networked control systems. *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, 2012.

[14] Keep Connected Cars Up to Date with FOTA (Firmware Over-the-Air) Technology. http://www.qnx.com/news/webseminars/fota.html.
[15] T. Flach and N. Mishra and L. Pedrosa and C. Riesz and R. Govindan. CarMA: Towards Personalized Automotive Tuning. *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011.
[16] M. Blanke, C. W. Frei, F. Kraus, R. J. Patton, and M. Staroswiecki. What is Fault Tolerant Control.
[17] R. Patton and P. Frank and R. Clark. *Fault Diagnosis in Dynamic Systems*. Prentice Hall, 1989.
[18] D.C. Fosth R.N. Clark and V.M. Walton. Detecting Instrument Malfunctions in Control Systems. *IEEE Transaction on Aerospace and Electronic Systems*, 1975.
[19] G. Welch and G. Bishop. An Introduction to the Kalman Filter. 2006.
[20] T. Pilutti, G. Ullsoy, and D. Hrovat. Vehicle Steering Intervention Through Differential Braking. *Proceedings of the American Control Conference*, 1995.
[21] nano-RK Sensor RTOS. http://nanork.org.
[22] J. W. S. Liu. *Real-Time Systems*. Pearson Education, 2000.
[23] B. Sinopoli and L. Schenato and M. Franceschetti and K. Poolla and M. I. Jordan and S. S. Sastry. Kalman Filtering With Intermittent Observations. *IEEE Transactions on Automatic Controls*, 49(9):1453–1464, 2004.
[24] AutoPlug: Open Architecture for Plug-n-Play Services. http://www.autoplug.org/.
[25] H. E. Tseng, B. Ashrafi, D. Madau, T. A. Brown, and D. Recker. The Development of Vehicle Stability Control at Ford. *IEEE Transactions on Mechatronics*, 4(3):223–234, 1999.
[26] CarSim Vehicle Simulator. http://www.carsim.com/.
[27] S. DSilva, P. Sundaram, and J.G. DAmbrosio. Co-Simulation Platform for Diagnostic Development of a Controlled Chassis System. *SAE World Conference*, 2006.
[28] Honda Recalls 936000 More Vehicles for Electrical and Software Fixes. http://spectrum.ieee.org/riskfactor/green-tech/advanced-cars/honda-recalls-936000-more-vehicles-for-electrical-and-software-fixes.
[29] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, and S. Savage. Experimental Security Analysis of a Modern Automobile. *IEEE Symposium on Security and Privacy*, 2010.
[30] A. A. Cardenas, S. Amin, and S. Sastry. Secure Control: Towards Survivable Cyber-Physical Systems. *28th International Conference on Distributed Computing Systems Workshops*, 2008.
[31] L. Parolini and B. Sinopoli and B. Krogh and Z. Wang . A Cyber-Physical-System Approach to Data Center Modeling and Control for Energy Efficiency. *Proceedings of the IEEE, special issue on Cyber-Physical Systems*, 2011.
[32] A. Teixeira and D. Huertas and H. Sandberg and K. H. Johansson. Cyber-Security and Safety Analysis of Cyber-Physical Systems. 2012.