Low Cost, Open-Source Platform to Enable Full-Sized Automated Vehicle Research

Austin Costley, Chase Kunz, Rajnikant Sharma, Member, IEEE, and Ryan Gerdes, Member, IEEE

Abstract—An open-source vehicle automation methodology, and platform for road vehicles (cars, and trucks) is presented. The platform hardware, and software, based on the Robot Operating System (ROS), are detailed. Two strategies are discussed for enabling the remote control of a vehicle (in this case, an electric 2013 Ford Focus). The first approach used digital filtering of Controller Area Network (CAN) messages. In the case of the test vehicle, this approach allowed for the control of acceleration from a tap-point on the CAN bus, and the OBD-II port. The second approach, based on the emulation of the analog output(s) of a vehicle's accelerator pedal, brake pedal, and steering torque sensors, is more generally applicable and, in the test vehicle, allowed for the full control vehicle acceleration, braking, and steering. To demonstrate the utility of the methodology, and platform for vehicle automation research, a very basic low-level velocity, and steering controllers are developed. Additionally a high level path following algorithm was developed. Finally, the system is validated experimentally.

Index Terms—Autonomous automobiles, autonomous vehicles, cyber-physical systems, intelligent vehicles, security, unmanned autonomous vehicles, vehicular automation.

I. INTRODUCTION

In recent years the automotive industry has been automating vehicle systems to aid drivers with features such as adaptive cruise control, lane keeping, and collision avoidance [1]. In more advanced systems, that are not commercially available, vehicles can drive themselves to user defined destinations [2]. Though these advanced driver aid systems are yet to emerge in production vehicles, research has been conducted for many years to help develop this technology [3]. The benefits of automated vehicles extend beyond convenience, and includes safer roadways, increased highway throughput, and reduced emissions.

Despite the positive effects of vehicle automation, there are possible drawbacks and considerations should be made regarding the safety and security of these automated systems. Hackers are constantly examining platforms in search of exploitable vulnerabilities and these automated features provide attack opportunities that were previously unavailable. For example, in [4] and [5], Miller and Valasek showed that it was possible to exploit the parking assist and lane keeping features of modern vehicles to gain limited control of acceleration and steering. In [6], Koscher et al. demonstrate the ability to disable the braking system of modern vehicles.

This work details the development of a low-cost experimental platform to enable automated vehicle research. This was accomplished by superseding internal sensor modes and injecting signals to (essentially) drive-by-wire components. The resulting platform can be used for research in the fields of automated vehicle control and dynamics and vehicle security.

The control of automated ground vehicles (AGVs) has been a subject of research since 1955 [7] and has naturally progressed to include commercially available passenger vehicles. In [3], Rajamani details various vehicle control systems, and modeling techniques. There have been several automated vehicle competitions held to further the research in this field. The DARPA Grand Challenge, for example, was started in 2004 to encourage researchers to develop off-road autonomous vehicle technology that could be used for military applications [8]; the challenge was repeated in 2005 [9]. In 2007 the challenge was altered to focus on urban driving environments [10]. The teams in these challenges started with an existing commercial vehicle and developed an automated system to complete the challenge. The Grand Cooperative Driving Challenge (GCDC) is another example of the advancement of automated driving through competition [11]. The purpose of the GCDC is to examine cooperative automated vehicle systems. Teams developed an automated vehicle to be used in cooperative challenges with other teams.

In contrast to the work by the teams in the DARPA Grand Challenge and GCDC, this work is open-source, low-cost, and has been completed without vendor support. One purpose of this work is to allow other researchers to perform vehicle automation tasks similar those in these challenges without needing support from industry.

Koscher et al. and Checkoway et al. in [12] and [6] showed that an attacker who can gain access to Electronic Control Units (ECU’s) could circumvent vehicle control and safety systems. They further demonstrated that an attacker could gain access...
to the controller area network (CAN) bus remotely to launch these attacks. Their results included the ability to have a car ignore a brake pedal press by the driver, a complete vehicle shut down, and complete control of the visual display. The attacks, however, did not allow for arbitrary control of acceleration, braking or steering, which is required for vehicle autonomy.

Miller and Valasek built on the work from Koscher et al. and Checkoway et al. in [4] and [5], which detail attacks on vehicle systems in a 2010 Ford Escape, 2010 Toyota Prius, and a 2014 Jeep Cherokee. They developed an extensive platform for attacking ECUs and exploiting driver assist systems. However, the attacks had limited scope for vehicle control. For example, an attack on the parking assist module was conducted on the 2010 Ford Escape, whereby vehicle steering could be controlled when the vehicle was traveling at under 5 mph, though would not work if the vehicle was moving faster. They also demonstrated the ability to cause vehicle acceleration under very specific conditions. In contrast, this work demonstrates the ability to cause arbitrary acceleration under any condition. In addition, using the approach presented here, the acceleration can be controlled from the OBD-II port, any tap point on the CAN bus, or by gaining access to an ECU. The work in this paper also demonstrates that a modern vehicle can be examined and reverse engineered, by a moderately knowledgeable attacker/engineer, to develop the automated system for a commercial vehicle.

The main focus of the paper is to develop a low-cost and open-source vehicle automation platform. To that end, our first goal was to enable remote control of the vehicle without adding hardware actuators. Two approaches were examined and prioritized in the following order: CAN message injection and sensor emulation. CAN message injection would attempt to take control of the vehicle by superseding modules connected to the CAN bus. Sensor emulation would supersede the user input sensors to manipulate the signals being sent to the CAN modules. A reverse engineering approach was taken to determine if the vehicle could be controlled through either of these methods.

For demonstration purposes, after remote control was enabled a basic control system was designed. First, model identification steps were taken to develop a model of vehicle dynamics due to user input. Low level feedback controllers were developed to allow control of meaningful vehicle states, and a high level controller was developed to coordinate the low level controllers to follow a desired path.

The main contributions of this work are as follows:

- Demonstration of an affordable, open-source, experimentally validated automation platform using the Robot Operating System (ROS) [13] that does not require actuation hardware.

- Two ways to manipulate control input to vehicles are examined: CAN message injection and sensor emulation.

- Software packages, hardware descriptions, and a methodological approach ready to be used to automate nearly any vehicle. The software packages are available at: https://github.com/rajnikant1010/EV Automation.

- Expose a vehicle vulnerability wherein a compromised ECU or other device connected to the CAN bus (including via the OBD-II port) could control vehicle acceleration.

The proposed platform is suitable for short, complex test tracks (≤ 1 km) and/or environments with moderate speed limits (≤ 30 mph), e.g., urban settings and most academic testing facilities. The platform is thus able to cover a diverse range of environments of interest to researchers while enabling testing of automation technologies that satisfy safety requirements for research participants.

II. ENABLING REMOTE CONTROL

Modern vehicles use a Controller Area Network (CAN) bus system for module-to-module communication [14]. Electronic Control Units (ECU’s) are the CAN modules that connect to the bus that send and receive information. A CAN module receives data from sensors, processes the data, and generates the appropriate CAN message to be broadcast on the bus using an analog-to-digital operation. We used the 2013 Ford Focus Electric Wiring Guide [15] and the Auto Repair Reference Center Research Database from EBSCOhost [16] to understand signal path and critical connections. Using these resources, the

| Sensor | CAN Module | CAN Message | CAN Arbitration ID |
|--------|------------|-------------|-------------------|
| Accelerator Pedal Position Sensor | Powertrain Control Module (PCM) | Accelerator Pedal Position | 0x204 |
| Brake Pedal Position Sensor | Automatic Braking System (ABS) | Brake Pedal Position | 0x7D |
| Steering Torque Sensor | Power Steering Control Module (PSCM) | Steering Torque | Unknown |
| Steering Wheel Angle Sensor | Steering Angle Sensor Module (SASM) | Steering Wheel Angle | 0x10 |

Fig. 1. Vehicle CAN bus and sensor architecture. Controller insertion type 1 filters CAN messages and replaces data with the message to be injected, and is represented by a filled black square. Controller insertion type 2 emulates sensor output signals, and are represented by filled black circles. The TCM controls the main drive motor of the electric vehicle and therefore actuates acceleration. The PSCM actuates the power steering motor. The ABS module actuates the hydraulic braking system.
team identified sensors and modules that could be used for vehicle control. Table I summarizes these findings.

The examination of this architecture led to the identification of the two possible controller insertion strategies, as shown in Fig. 1. First, the CAN lines between the control module and the CAN bus could be cut, and a controller could be inserted to intercept and change messages being sent from the target control module. Second, the analog signal wires from the target sensor could be cut, and the controller could be inserted between the sensor and the control module. In either strategy, the controller would insert spurious data into the system to control the vehicle. The details and results of these two approaches are discussed in the following subsections.

In order to successfully implement the first controller insertion strategy and take advantage of the information on the vehicle CAN bus, the Vector CANalyzer [17] system was used for the initial CAN message identification process. This system was used to isolate useful CAN messages and characterize the message. Alternatively less expensive systems could be used for this process, such as the Peak Systems PCAN [18]. An open-source solution for monitoring CAN traffic with the open-source software accompanying this work. Further discussion on the use of the PCAN device can be found in Section V-A. Another alternative is to use a microcontroller with a CAN bus interface module to monitor and report CAN traffic [19].

A. CAN Message Injection

A platform for injecting CAN messages was developed using the TI TM4C129XL microcontroller evaluation kit [19] and TI CAN transceivers [20]. The platform would connect to the CAN bus using the first controller insertion point, between the target module and the bus. The microcontroller was programmed to record and playback CAN traffic. More specifically, the microcontroller would receive the output from the control module and store it in memory. Upon user request, the output from the control module could be injected on the CAN bus. This platform was used to determine which messages, organized by arbitration ID, were generated by the target control module. Using information from the Auto Repair Center Research Database, it was determined that the Powertrain Control Module (PCM) sent a CAN message to the Transmission Control Module (TCM) reporting the accelerator pedal position. In addition, the PCM was the only control module that received the sensor output from the TCM. It was determined that the sensor outputs two DC voltages similar to the output of potentiometers. Fig. 2(a) shows successful CAN injections of the two signals. It is seen that $V_1 \approx 2V_2$.

The investigation of the braking system concluded that a CAN bus message about pedal position would not actuate the hydraulic braking system. The pedal signal is sent directly to the Automatic Braking System (ABS) CAN module, which is the only CAN module on the vehicle that is connected to the hydraulic brake lines. From this it was concluded that braking could not be fully controlled through the CAN bus.

The 2013 Ford Focus EV has an option to include park assist [21]. Though our specific vehicle did not include this option, it was determined that the Electric Power Assisted Steering (EPAS) system had the same part number and motor as the EPAS system in a vehicle with the park assist feature. This meant that the power steering motor would be powerful enough to turn the wheel at low speeds, and by extension, any speed. The Power Steering Control Module (PSCM) receives inputs from the CAN bus and the steering torque sensor located at the base of the steering column. The torque sensor uses a torsion bar to determine the amount of torque being applied by the driver, which is used by the PSCM to determine how much assist the power steering motor should provide. Simply, for a given torque input, less assistance would be provided by the EPAS system at higher speeds. Similar to the brake system, the input of interest is sent from a sensor to the module that performs the desired action. This led to the conclusion that the control of steering would have to be controlled through torque sensor input, and not through the CAN bus. In addition, the work by Miller and Valasek [4], [5] identifies the shortcomings of exploiting the park assist feature, namely, the park assist feature will cease to control the vehicle if the speed threshold is exceeded.

B. Sensor Emulation

The CAN bus injection method was unable to control braking and steering, so the sensor emulation method was explored. The accelerator pedal position sensor was analyzed to control vehicle acceleration, the brake pedal position sensor was analyzed to control the braking system, and the steering torque sensor was analyzed to control the vehicle steering. Fig. 3(a) shows these sensors and the following paragraphs discuss the analysis. The accelerator pedal position (APP) sensor is located at the top of the accelerator pedal. It was determined that the sensor outputs two DC voltages similar to the output of potentiometers. Fig. 4(a) shows the voltage levels of the two output signals in response to a pedal press. The third signal on the graph is a multiplier that relates the two signals. It is seen that $V_1 \approx 2V_2$.

The brake pedal position (BPP) sensor is located at the top of the brake pedal. It was determined that the BPP outputs two PWM signals instead of DC voltage levels. When the brake
Fig. 2. CAN throttle message injection from controller insertion point 1 and through OBD-II port, with resulting vehicle speed. All values read from CAN bus and represented in hexadecimal format. (a) Ramp injection from controller insertion point 1. (b) Step injection from controller insertion point 1. (c) Ramp injection through OBD-II port. The CAN bus was monitored from the OBD-II port, and the injected throttle message was broadcast on the CAN bus immediately following the reception of vehicle throttle message.

Fig. 3. (a) The physical sensors emulated for vehicle control. Top: Steering rack for 2013 Ford Focus EV, the steering torque sensor is located at the base of the steering column and measures torque from driver. Bottom Left: The APP sensor located at the top of the accelerator pedal. Bottom Right: The BPP sensor that is usually mounted behind the brake pedal assembly and measures the brake pedal press. (b) Aerial view of the Electric Vehicle Roadway and Research Facility (EVR) at Utah State University. (c) High-level system block diagram. Shows low-level control loop for lateral and longitudinal control and high-level differential flatness path following feedback loop.

Fig. 4. (a) Accelerator pedal position sensor output. Two analog voltage signals related by $V_1 = 2V_2$. (b) Brake pedal position sensor output signals. Signal 1: 89% resting duty cycle at 533 Hz. Signal 2: 11% resting duty cycle at 482 Hz. (c) Steering torque sensor output signals. 50% resting duty cycles at 2.15 kHz.

Similar to the BPP, the steering torque sensor outputs two PWM signals on the signal wires, where both signals settle at 50% duty cycle when no torque is applied. Both signals have a frequency of 2.15 kHz. Similar to the brake PWM signals, the duty cycles always add to 100%, and the direction that the steering wheel is being turned determines which signal’s duty cycle increases and which signal’s duty cycle decreases. Fig. 4(c) shows the two steering PWM signals.

C. Safety and Security

In 1996, the OBD-II (On-Board Diagnostics) specification was required to be implemented on any new vehicle sold in
the United States [23]. This specification gives owners and technicians the ability to diagnose issues on the vehicle. The specification standardized connectors, message formats, and frequencies. The OBD-II port on the 2013 Ford Focus EV connects to the EV-HS CAN bus, which is the same bus that the throttle message is sent from the PCM to the TCM. An attack platform was developed to inject arbitrary throttle messages through the diagnostics port. This attack method was important because, if successful, it would demonstrate that the acceleration of the vehicle could be controlled with limited intrusion. This differed from the approach in Section II-A, as it does not require access to the target module, or that the CAN wires be cut and re-routed. Instead, this platform could be plugged into the OBD-II port and monitor the bus for the target message arbitration ID. Also, it would show that if an attacker was able to inject messages from any module on the EV-HS CAN bus, then arbitrary vehicle acceleration could be caused. This would stand in contrast to the findings in [4], [5], [6], [12], where the acceleration of the vehicle could only be controlled under specific preconditions, and required intrusive access to the CAN bus.

The platform was connected to the CAN bus through the OBD-II port (other points on the bus could be used, as well) and monitored the traffic on the bus. The user determined a target message, in this case, the throttle message, and provided that message arbitration ID to the system. In Section II-A, it was determined that the throttle message is included in the data frame associated with arbitration ID 0x11 A, and is broadcast at 10 Hz. The platform waited until a message was received with the corresponding arbitration ID, and would replace throttle message data with an arbitrary throttle command value. The platform was designed to only alter the parts of the message that relate to the throttle control. The inserted message would be sent 250 \(\mu s\) after the actual message, leaving 9.75 ms for the inserted message to be received and processed by the TCM. This allowed the inserted message to dominate the period and cause the vehicle to accelerate. Fig. 2(c) shows the successful ramp injection through the OBD-II port and the resulting vehicle speed. Thus confirming the hypothesis that vehicle acceleration can be caused by injecting CAN messages through the OBD-II port, and therefore, could be caused at any other point on the bus.

These results demonstrate a CAN bus security concern. If an attacker were able to access the CAN bus, physically, or by compromising another ECU, they would be able to affect the acceleration of the vehicle without causing any errors. This vulnerability could be exploited by remote access to a CAN module (not explored in this work) or through physical access to the OBD-II port. Regardless of the access approach, the driver is able to stop the unwanted acceleration by pressing the brake pedal, however, other works indicate that it is possible to make the vehicle ignore braking requests [4]–[6], [12]. We present two remediation strategies that could be employed to help protect against this vulnerability. First, a simple change in the acceleration system architecture, such that the APP sensor connects directly to the TCM, which is the actuating module. This would remove the need of a throttle message to be sent from the PCM to the TCM and effectively remove the attack surface. The second approach is through device fingerprinting for both the digital and analog signals [24], [25]. This would allow the receiving module to authenticate the transmitting module, and prevent this type of attack.

### III. Low Level Control

A simple overview of the control structure for the automated vehicle platform is shown in Fig. 3(c). The high level controller plans the path and provides the desired vehicle speed, \(v_{\text{desired}}\), and desired steering wheel angle, \(\theta_{\text{desired}}\), discussion on the high level controller can be found in Section IV. The low level controllers discussed in this section are the inner loops that control vehicle speed and steering wheel angle. The vehicle commands are \(\tau_{\text{cmd}}\), \(APP_{\text{cmd}}\), and \(BPP_{\text{cmd}}\), and represent, steering torque, APP, and brake pedal position, respectively. The following subsections review the model identification approach, and low level controller design process for the Ford Focus.

#### A. Longitudinal Model

The longitudinal characteristics of the vehicle are affected by the APP sensor, and the BPP sensor. These two systems were tested and modeled separately, then implemented together as a complete longitudinal model. For the APP sensor system identification, the vehicle was placed on a dynamometer [26] and step inputs were initiated on the APP sensor from 4% to 15% at increments of 1%. as shown in Fig. 5(a). It was observed that for a given APP percentage the vehicle would eventually settle at a specific speed. For the operating conditions considered, which we believe are broadly representative of those available to most researchers, the relationship between APP and speed can be described by a first order transfer function that varies by speed. The general equation for a first order transfer function, \(G(s)\), can be represented by

\[
G(s) = \frac{K}{\tau s + 1}. \tag{1}
\]

Where \(K\) is the constant or equation that relates APP to vehicle speed, and \(\tau\) is the system time constant. The system identification showed that \(K\) is a function of APP: Thus, the equation for \(K(\text{APP})\) was derived from a linear fit of the a scatter plot of max speeds from the step input, as shown in Fig. 5(b), and given by \(K(\text{APP}) = 3.65APP - 9.7\). The time constant \(\tau\) that best fit the desired operating speeds (15–25 mph) was chosen. Fig. 5(c) shows the time constants for varying APP percentages. The time constant for the accelerator pedal input was chosen to be 7 seconds, as this best represented the system response for the nominal operating conditions.

For BPP system identification, the vehicle was driven in a large, flat, asphalt area at speeds ranging from 5 to 25 mph at 5 mph increments. The vehicle was accelerated to the desired speed by a driver. Once the vehicle obtained the desired speed, step inputs were initiated ranging from 5% to 50% of BPP percentage at increments of 5% for each speed value. The speed data seemed to show a consistent rate of change for a given BPP percentage. Fig. 5(d) shows the vehicle deceleration due to a braking event. It was observed that the settling value for
Fig. 5. (a) APP step response for 5%, 10%, and 15% pedal presses. (b) Vehicle settling speeds for given APP step input percentages. (c) Time constants for given APP step input percentages. (d) Deceleration rate for BPP step input of 15%.

Fig. 6. (a) Vehicle deceleration for BPP percentages at a variety of speeds. (b) Average deceleration settling rates due to BPP step input percentages. (c) Steering torque step response for 58%, 60%, 62%, and 64% duty cycles at a vehicle speed of 25 mph. (d) Steering torque step inputs for 58%, 60%, 62%, and 64% duty cycles at a vehicle speed of 15 mph.

the deceleration rate was consistent for a given BPP percentage and varying speeds, which concluded that the longitudinal model was independent of current vehicle speed. This speed independence can be seen in Fig. 6(a) where each line shows the deceleration rate for a given BPP percentage. At low BPP values the lines converge meaning that deceleration is unaffected by very small brake pedal percentages. However, at higher brake pedal percentages the lines show distinct deceleration rates regardless of the vehicle speed. To show the relationship between BPP percentage and deceleration, an average was taken for each BPP value across each of the speeds. The result of this operation is shown in Fig. 6(b).

Similar to the APP model, the relationship between BPP and deceleration could be described by a first order transfer function with a polynomial gain based on pedal position. After analyzing the deceleration curves at different BPP percentages and for different speeds the system time constant, $\tau$, was calculated to be 0.3 seconds. The equation that relates BPP to deceleration was determined by finding a curve fit algorithm for the curve in Fig. 6(b). This would result in an equation that would provide a BPP percentage for a desired deceleration rate. The equation for $K(BPP)$ is given by $K(BPP) = -0.0018BPP^2 + 0.029BPP - 0.3768$. This equation is used to describe $K$ from the general first order transfer function equation.

B. Lateral Model

The lateral model of the vehicle was determined by step response analysis. The model relates an input from the steering torque sensor to changes in the steering wheel angle. As discussed in Section II-B, the torque sensor measures the torque applied by the driver, and sends that information to the PSCM. The PSCM activates the power assist motor that connects to the steering rack, and moves the wheels. The steering wheel angle is measured by a sensor in the steering wheel and output on the CAN bus at a high level of precision.

Step inputs were initiated on the steering torque duty cycle signal ranging from 50% to 63% at 1% increments. Tests were performed at a large, flat, asphalt area with vehicle speeds ranging from 5 mph to 25 mph. Fig. 6(c) shows the results of the step input tests performed at 25 mph. It was observed that a general first order transfer function could be used to describe the relationship between steering torque duty cycle and steering wheel angle. However, at lower speeds and higher torque values this observation is not valid. Fig. 6(d) shows the step response of the steering system at 15 mph. At the higher torque values the steering wheel angles do not settle to a consistent steering wheel angle. It was also observed that the settling angles for a given steering torque duty cycle are not consistent for varying speeds. Therefore, the lateral model identification is speed dependent and would require a speed dependent limit on the steering torque duty cycle. Providing these characteristics, the system can still be modeled as first order transfer function for a given speed.

The steering data was analyzed in order to determine the gain, $K$, and the time constant, $\tau$. Time constants were calculated for each step input response and for each speed. Fig. 7(a) Bottom shows the time constants for given steering torque duty cycle. Each of the lines indicates the speed at which the test was performed. It can be seen that at low speeds and low duty cycles the time constants are not consistent. But at
higher speeds the inconsistencies lessen. A time constant, $\tau$, of 0.2 seconds was chosen to optimize for typical vehicle operation.

Since the lateral system was found to be speed dependent, the gain equation $K$ must also be speed dependent. The step input tests were performed at 5 mph increments so a gain equation $K$ would be found for each speed value. These gain equations relate steering torque duty cycle to steering wheel angle. Fig. 7(a) shows the settling angles for varying steering torque duty cycles when the vehicle was traveling at 25 mph. A curve fit approximation was completed for this data set, and a solution was determined by solving the given equation. For this data set, the given equation for $K$ is $K(s_t) = 59.4s_t^2 - 6802.7s_t + 195084.5$, where $s_t$ is the steering torque duty cycle. Figs. 6(c) and 6(d) show the step response of the vehicle due to steering torque input signals. The graphs do not include step input values below 58% because the step responses at such values had little effect on the steering wheel angle. This exposed a deadband in the response from the steering torque sensor input to the steering wheel angle. A deadband compensation algorithm was implemented to mitigate the effects of this non-linearity. For the deadband on the 2013 Ford Focus EV, the upper and lower limits were 55% and 45%, and the maximum and minimum values for the torque signal were 64% and 37%, respectively.

Low-level control loops were designed to control vehicle speed and steering wheel angle. The desired speed and desired steering wheel angle would be input to the low-level control loops from a user or high-level controller. The low-level longitudinal controller interfaced with the accelerator and brake pedals to effect vehicle speed. A separate loop was designed for each vehicle input, and switching logic was used to choose whether the acceleration or brake loop would be used. The low-level lateral controller would receive the desired steering wheel angle and determine the appropriate input to the steering torque sensor to achieve the desired angle. A Proportional Integral (PI) Feedback Controller was implemented for longitudinal and lateral control.

IV. HIGH LEVEL CONTROL

The high level controller was designed to take a desired trajectory or path, and provide appropriate inputs for the low-level controllers. This platform was to be used on a ground vehicle to track a desired trajectory and was determined to be differentially flat [27] for the chosen states. Therefore, a simple high level controller using the properties of a differentially flat system, and state feedback control were chosen. A virtual target scheme was selected for path and trajectory control. The virtual target data was gathered by driving the vehicle around the test track and recording the RTK-GPS data at 10 Hz with 2 cm resolution. The Linear Quadratic Regulator (LQR) method was used to find the optimal desired steering wheel angle and the desired velocity that are sent to the low level controllers discussed in Section III.

V. PLATFORM OVERVIEW

A versatile and robust platform is required to enable full-sized autonomous vehicle research. The platform was designed to enable vehicle automation for both the CAN injection and the sensor emulation approaches discussed in Section II. For the CAN injection approach the platform was able to monitor the CAN bus and inject the desired packets, whereas for the sensor emulation approach the platform required access to the output lines of the sensors to be emulated. In order to proceed to autonomy the platform required the ability to sense the environment, determine vehicle location, communicate with the vehicle, and monitor the CAN bus. A computer running Ubuntu and the Robot Operating System (ROS) was used to communicate between the platform architectures. The computer was connected to a microcontroller to allow communication with the vehicle. Fig. 8 shows a diagram of the autonomous system, including the ROS software architecture, hardware connections to devices, and the vehicle interfacing hardware.

A. Interfacing Architecture

Interfacing devices are critical to the success of vehicle automation as they provide a way to send commands to the vehicle, and monitor the vehicle for feedback. A PCB (shown in Fig. 9(a)) was designed to provide a connection between the microcontroller and the vehicle. The following subsections discuss the microcontroller and associated hardware, and the other interfacing devices used for this platform.

1) Microcontroller and Associated Hardware: The TI TM4C129XL evaluation kit was the chosen microcontroller platform because it offered multiple CAN bus interfaces allowing for a combination of CAN injection and sensor emulation from the same board [19]. The microcontroller receives input from the computer through a UART module.

PWM signals are generated and appropriate DC voltage levels are determined for vehicle input. The signals are terminated at solid state relays that select either the original vehicle signal, or the generated signal to be sent to the vehicle. The user determines which signal is sent based on a mode switch input to the microcontroller.

The sensor emulation approach requires four PWM signals to be generated by the microcontroller. The signals are passed
was chosen to monitor CAN traffic [18]. The PCAN device can across the custom PCB. The UPS is shown in Fig. 9(c).

NewMar UPS also has an internal backup battery. The voltage level is stepped down to and stable voltage levels for the circuit operations [28]. The circuit is provided by a NewMar DC Uninterruptible Power Supply the power returns the vehicle to Manual Mode. Power to the circuit. The vehicle’s original sensor signals are connected to the normally opened terminals of solid state relays.

Another key feature of the PCB is the safety circuit. Next to the driver there is a mode switch and an Emergency Stop button. The mode switch allows the driver to switch between Manual Driving Mode and Autonomous Mode. The power and solid state relay control signals are routed to the front of the vehicle so the safety driver can switch between operating modes or press the Emergency stop button to prevent power from reaching the circuit. The vehicle’s original sensor signals are connected to the normally closed terminals of the solid state relays, so removing the power returns the vehicle to Manual Mode. Power to the circuit is provided by a NewMar DC Uninterruptible Power Supply (UPS) which connects to the 12 V car battery, and provides safe and stable voltage levels for the circuit operations [28]. The NewMar UPS also has an internal backup battery. The voltage level is stepped down to ±8 V, 5 V and 3.3 V, and distributed across the custom PCB. The UPS is shown in Fig. 9(c).

2) Other Interface Devices: The Peak Systems PCAN device was chosen to monitor CAN traffic [18]. The PCAN device can connect directly to the vehicle’s OBD-II port, and provides serial output over USB. A picture of the PCAN device is shown in Fig. 9(b). Every message on the connected CAN bus is received and sent serially to the computer. The user can determine which CAN data packets are important to operation, and ignore the rest. One approach to determine necessary CAN data packets is detailed in Section II. Instead of using the CANalyzer system to monitor CAN traffic, the PCAN device can be used to record CAN traffic for a desired event (e.g. vehicle acceleration). The CAN data can be replaced section by section until a message or set of messages is isolated. Additional information on the use of the PCAN device for system feedback is given in Section V-C2.

A USB-to-seral device was used between the microcontroller and computer to enable communication. The control system determines the appropriate inputs to the vehicle and sends the commands to the microcontroller. The device receives the signal from the USB port and sends it to a UART module on the microcontroller. More information about the microcontroller and control system is given in Section V-C. The TI SN65HVD230 CAN Transceiver Breakout Board [20], [29] was used to connect the microcontroller to the CAN bus. This board provided a direct connection with the vehicle CAN bus that can be used for monitoring and injection.

B. Sensing Architecture

The Swiftnav Piksi RTK-GPS unit [30], [31] was chosen to provide position and velocity estimates for the vehicle. A picture of the Swiftnav Piksi unit is shown in Fig. 9(b). Swiftnav provides an inexpensive, open-source RTK-GPS solution that provides GPS measurements at 10 Hz and ±2 cm accuracy. In order to achieve such high accuracies the system must have a base station with an RTK-GPS unit, and a second unit mounted to the vehicle. The two units communicate with radio transceivers at 955 MHz. The unit mounted on the car connects to a computer over USB and provides position and velocity data. The base station simply needs a 5 V power supply. The 2013 Ford Focus EV has an array of sensors on the vehicle that monitor everything from wheel speed to tire pressure. However, the vehicle does not have high precision wheel encoders, an RTK-GPS receiver, or inertial measurement sensors (IMU’s) that could be useful for vehicle automation. The sensor data is typically received by a module and sent on the CAN bus. Using the PCAN device described in Section V-A, the on-board sensor information can be provided to the rest of the automation platform. For autonomous driving, the accelerator pedal position sensor, brake pedal position sensor, steering wheel angle sensor, and vehicle speed data are used for feedback in the low-level controllers.

The sensor data broadcast on the CAN bus does not provide the information in empirical units, and sometimes the data is masked with other signals. An important aspect to the sensing architecture is the conversion from CAN bus messages to useful units. These conversions could be found experimentally for each message found on the CAN bus, but for the purposes of this platform the vehicle speed was the only message converted to empirical units (MPH). The vehicle speed is found on the
message with arbitration ID \(0\times75\) on bytes 7 and 8, and is represented by a 16-bit value where byte 7 is the upper byte and byte 8 is the lower byte. When the vehicle was not moving the vehicle speed was represented as \(0\timesB0D4\) on the CAN bus. The decimal representation of this constant, 45,268, is subtracted from the 16-bit speed value to align the 0 MPH value. The vehicle was driven with the RTK-GPS units to provide a reference for the vehicle speed, and it was determined that the CAN value would then need to be divided by 54 to achieve an accurate measure of speed. This process is summarized by

\[
\nu_{\text{mph}} = \frac{b_7 << 8 + b_8 - 45268}{54},
\]

where \(b_7\) and \(b_8\) are the integer representations of bytes 7 and 8 from the CAN message with arbitration ID \(0\times75\), and the \(<<\) operator represents a left bit shift.

C. Computational Architecture

The computational architecture includes the code required to combine sensor information, controller commands, and prepare command insertion. The two computational platforms used in this system are the microcontroller and ROS. These platforms are discussed in the following sections and code for these platforms can be found at https://github.com/rajnikant1010/EVAutomation.

1) Microcontroller Software: The code for the TI TM4C129XL was written in C and took advantage of the built-in functionality of the TivaWare Peripheral Driver Library from Texas Instruments [32]. Table II shows the peripherals used and their functionality. The following paragraphs discuss the microcontroller code.

The microcontroller receives a serial packet from the computer in the form shown in Fig. 10. The first byte is always \(0\timesFA\), the second byte gives the number of bytes in the payload, the payload contains the steering, braking and acceleration commands to be sent to the vehicle, and the last two bytes is a 16-bit Cyclic Redundancy Check (CRC) using the CRC16-CCITT algorithm to ensure data integrity. Once received, the CRC is calculated to ensure correct data, and the payload values are stored in appropriate variables. All input commands are normalized between zero and one. For PWM signals the normalized value represents the duty cycle of the signal, where 0.5 represents 50% duty cycle.

2) ROS Architecture: The ROS architecture shown in Fig. 8 consists of a series of packages, nodes, and topics [33]. There are four packages used for this project were swiftnav_piksi (GPS), focus_serial (serial communication), pcan (CAN interface), and focus_control (high- and low-level control).

VI. EXPERIMENTAL RESULTS

Experiments were conducted to determine the results of the autonomous vehicle platform. A video of the system in operation and the development process can be found at https://youtu.be/7ohWIwb6KfM. The low level controllers were tested with given desired steering angles and velocities.

| Port | Pin | Type | Purpose            |
|------|-----|------|--------------------|
| GPIO_P | 2 | PWM | Steering signal 1  |
| GPIO_P | 3 | PWM | Steering signal 2  |
| GPIO_P | 1 | PWM | Brake signal 1     |
| GPIO_G | 1 | PWM | Brake signal 2     |
| GPIO_K (I2C4) | 6 | I2C_SCL | Acceleration I2C SCL line |
| GPIO_K (I2C4) | 7 | I2C_SDA | Acceleration I2C SDA line |
| GPIO_C | 4 | Logic | Mode select signal from user |
| GPIO_C | 5 | Logic | Mode signal to system |

| 0xFA | # of bytes (n) | Payload: Steering Torque, BPP, and APP Commands | CRC16-CCITT |
|-------|---------------|-----------------------------------------------|------------|
| 1-byte | 1-byte | n-bytes | 2-bytes |

Fig. 10. Serial message structure for communication with the microcontroller. The serial communication sends commands to the vehicle emulating the APP, BPP, and steering torque sensors. The second byte indicates the number of bytes in the payload, \(n\).
The low level steering controller was improved by implementing a deadband compensation algorithm. After the low level controllers were verified through testing, experiments were conducted for full vehicle automation by including the high-level path following controller. The low level controller was tested using a step input and a graph of the result can be seen in Fig. 11(a). The y-axis is the steering wheel angle as represented by a Hex value on the CAN bus. A desired steering wheel angle of 0x7D0 was used as an input to the controller. The step response had a maximum value of 0x891, representing an overshoot of 9.65%. The resulting time constant of the system was 1.86 seconds. The desired behavior was for the system to be critically damped and have a time constant of 0.33 seconds. After implementing deadband compensation, the lateral controller improved. Fig. 11(b) shows the step response of the lateral controller with deadband compensation. The maximum value for this response is 2113, which represents a 5.65% overshoot, and has a time constant of 1.1 seconds.

The result of the longitudinal controller during autonomous driving is shown in Fig. 11(d). The velocity error is shown in Fig. 12, where the average error for the autonomous test was 0.34 mph (0.151 m/s). The high level controller was tested on the 2013 Ford Focus EV. A recorded data set of the vehicle being driven around the track was used as a virtual target for the differential flatness algorithms discussed in Section IV. The lateral error was calculated using a reference point at approximately the middle of the track, with the distance between the reference path and the vehicle’s path from the reference point calculated using the Euclidean distance. As we are performing path following and not trajectory tracking, we have omitted longitudinal error.

A single lap autonomous test was performed with the high- and low-level controllers, and the data was recorded. The trajectory is shown in Fig. 11(c), and the velocity plot is shown in Fig. 11(d). The velocity and lateral path error plots are shown in Fig. 12, where the average lateral error was −0.08 meters and average absolute error 0.39 meters, which are commensurate with existing work [34]. We note that the large initial error is due to the fact that the vehicle was not initially positioned on the reference path. A 34.9% reduction in path error was observed if the vehicle was permitted to immediately perform a second lap of automated driving.

The authors would like to thank the Utah State University (USU) EV Automation Team and the USU Power Electronics Lab (UPEL).

ACKNOWLEDGMENT

The authors would like to thank the Utah State University (USU) EV Automation Team and the USU Power Electronics Lab (UPEL).

REFERENCES

[1] M. Casey, “Want a self-driving car? look on the driveway,” The Fortune, Dec. 6, 2014. [Online]. Available: https://fortune.com/2014/12/06/autonomous-vehicle-revolution/
[2] E. Guizzo, “How Google’s self-driving car works,” IEEE Spectr. Online, vol. 18, no. 7, pp. 1132–1141, 2011.
Austin Costley received the M.S. degree in electrical engineering at Black Hat USA in 2017 from Utah State University, where he is currently working toward the Ph.D. degree in electrical engineering. He is currently a GNC Engineer at Autonomous Systems, Inc. He works on the development of algorithms for autonomous vehicles. His research interests include GPS-denied navigation and autonomous vehicle localization and control.

Chase Kunz received the M.S. degree in electrical engineering from Utah State University in May 2017. He is a Robotics Perception Engineer with Inertial Sense, Inc. His primary research interests include computer vision and unmanned vehicle localization and control.

Rajnikant Sharma (Member, IEEE) received the B.E. degree in electrical engineering from the University of Rajasthan, Jaipur, India, in 2003, the M.E. degree in aerospace engineering from the Indian Institute of Science, Bangalore, India, in 2005, and the Ph.D. degree in electrical engineering from the Brigham Young University, Provo, UT, USA, in 2011. He is currently an Assistant Professor with the Aerospace Engineering and Engineering Mechanics Department, University of Cincinnati. From August 2013 to December 2016, he was with Electrical and Computer Engineering Department, Utah State University, Logan, UT, USA, where he was an Assistant Professor. From 2011–2013, he was a Postdoctoral Fellow with the Academy Center for UAS Research, US Air Force Academy, Colorado. From 2005 to 2007, he was a Scientist B with the Center for Airborne Systems, Defense Research and Development Organization, Ministry of Defense, Bangalore, India. His primary research interest include guidance, navigation, and control of unmanned aerial vehicles and multiple vehicle coordination, control, and localization. He is a member of the AIAA.

Ryan Gerdes (Member, IEEE) received the Ph.D. degree in electrical engineering for his work on device fingerprinting from Iowa State University in 2011. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Virginia Tech. From 2011 to 2016, he was an Assistant Professor with Utah State University. His research interests include cyber-physical systems security, with an emphasis on the operation of autonomous systems in unknown, uncertain, and adversarial environments, device fingerprinting, embedded systems security, sensor security, controls security, and cybersecurity.