Vehicle immobilization device via OBD-II port CAN packet injection

Annamalai Chockalingam, Farez Halim, Redge Santillan, and Mohamad Yassin
Group 7, Electrical Engineering Capstone Project, University of Alberta

***Abstract*—This paper outlines the design and implementation of the Vehicle Immobilization Device. The device monitors a vehicle's speed through On-Board Diagnostics (OBD-II) queries, and acceleration through an ADXL345 digital micro-electro-mechanical system (MEMS) accelerometer. Upon detection of a collision, defined as a 3g spike in acceleration in the x- or y-direction within a 0.5 second window, the device immobilizes the vehicle by injecting CAN packets to Electronic Control Units (ECUs) within the vehicle. These packets perform actions to keep a vehicle stationary. The action can involve commanding the brakes to actuate (preferred method), or shifting the transmission to 'Park'. To disable immobilization, an override is sent through Bluetooth using an Android app, to the device to halt Controller Area Network (CAN) message injection. The preliminary design of the final product is housed inside a dongle of length of 132 mm, width of 65 mm, and depth of 25 mm. This dongle shall attach to the vehicle's OBD-II port. A proof-of-concept device was built and tested with a custom-developed ECU simulator. The proof-of-concept was able to control the CAN-connected ECU simulator to perform an action to indicate an immobilized state, as well as reverse this "immobilization" through use of an Android app.**

I. Introduction

In 2015, over 140,000 accidents each causing over $2000 in damage were reported in Alberta. Approximately 31.2% of the reported accidents, were caused by vehicles following-too-closely, usually resulting in front-end collisions. Many of these accidents cause significant injuries and fatalities to all involved parties, including passengers, drivers, and bystanders. [1] Subsequently, insurance claims for accidents of this nature have been rising, and these claims are complicated, especially when vehicles leave the scene of the accident.

The Vehicle Immobilization Device (VID) is designed to address this issue by immobilizing a vehicle after an accident has occurred; effectively "freezing-the-scene" of the accident. The initial target end-users of this product are fleet-management organizations, such as vehicle rental organizations. This product will help mitigate risks associated with vehicle investment, insurance, maintenance of vehicle fleet, while gaining more control of the fleet, and ensuring compliance to government legislation and insurance policies. The requirements for the VID are outlined Table I.

TABLE I
Project requirements

Requirement	Brief description
Accident detection	Accurately identify a front end collision
Data collection	Sensors for accident detection must sample at a suitable rate such that accidents are not missed and false positives are not triggered
Easy to install	Installation of the device should not require significant cost or effort and should not be intrusive
Identify safe immobilization conditions	Device must attempt to immobilize only if the vehicle speed is less than 5km/h, for safety reasons
Immobilize the vehicle	Device must perform a reversible action that will effectively prevent the vehicle from further moving. The vehicle's climate control should remain functional during immobilization to ensure safety of vehicle operator in harsher climates.
No permanent changes to vehicle	The immobilization of the vehicle must only be temporary; no permanent change to regular operation of the vehicle should be made. The device must not interfere functionality of the engine, transmission, and other vehicle systems.
Disabling of the immobilization	Device must include functionality for reversal of immobilization only through an authorized party in a secure manner
Status indication	Should have indication representing its state to the end-user
Long lasting	Should not require maintenance at a high frequency and should have a lifetime of several years
Reliable	Should not act on false positives (potholes, etc)
Scalable	Should work on most late-model passenger vehicles and trucks
Tamper-proof	Should prevent the possibility of vehicle operator to tamper with the device and make it dysfunctional.
Able to function in Alberta climate	The device should have components rated for operation within the wide temperature ranges in the Alberta climate (-40 °C to 40 °C)

II. Background Information

A. Modern vehicles and the Controller Area Network

Modern vehicles are controlled via electronic signals and actuated mechanically. To actuate these mechanical components, there are various computers located throughout the vehicle. These computers are called ECUs, and each one is responsible for one function. Some examples of ECUs include: Transmission Control Unit (TCU), Braking Control Module (BCM/ABS), Electronic Power Steering Unit (PSCU). The network architecture within the vehicle is based upon distributed computing (each ECU is a node), and the fabric connecting these various ECUs is referred to as the Controller Area Network (CAN)[2]. A typical CAN architecture can be seen in Figure 1.

B. Vehicle On-Board Diagnostic Port

All vehicles manufactured in North America after 1996, must come with an OBD-II port (On-Board Diagnostics Port), often located close to the steering wheel and dashboard of the vehicle. OBD-II is a 16-pin hardware port, and is typically used by mechanics to retrieve the status of subsystems within

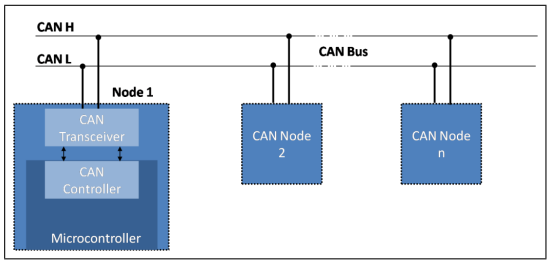

Fig. 1. The CAN bus architecture consists of a 2-wire (CAN Hi/Low) hardware based communication messaging protocol utilizing differential signaling on the two wires. A CAN transceiver is located on each node to convert this differential signal to CAN Receive (Rx) and Transmit (Tx) signals, which are then connected to a CAN controller which converts the CAN Rx/Tx to a typical MCU protocol such as SPI.[3]

a vehicle. The port is connected to various communication buses, including the CAN bus. [4][5]

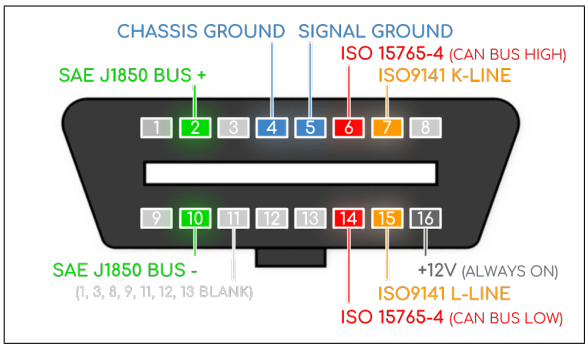

Fig. 2. Standard OBD-II pinout/connection diagram.[5]

III. Design

A. Overview

The three critical device functions were identified as follows, with the chosen method to satisfy that requirement:

- Detection of front-end collision achieved through measuring acceleration at a 100 Hz sampling rate and checking for a 3g spike in the x- or y-direction within a 0.5 second window.
- Immobilization of the vehicle is realized through injection of packets, in CAN format, through the OBD-II port to mimic brake actuation packets, thereby applying the brakes of the vehicle through an electrical signal and preventing the vehicle from moving.
- Disabling the device, i.e. reversing the immobilization, is achieved through a smartphone app connected to a realtime database backend containing unique keys for each device.

The major components in the design are as follows, and is illustrated in Figure 3.

- ADXL345 Digital MEMS Accelerometer
- Raspberry Pi Zero W for processing
- STN1110 OBD-II to UART Interpreter integrated circuit
- Android app for disabling through secure means

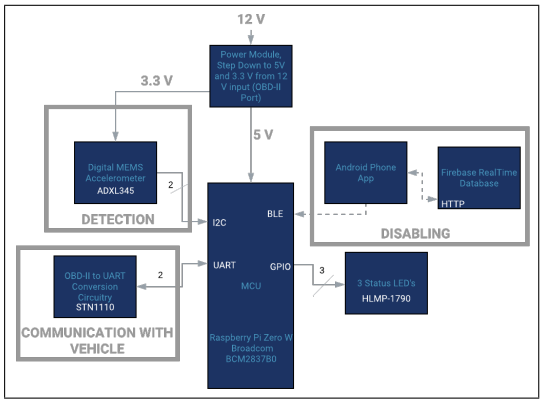

Fig. 3. High-level block diagram of the VID

- Firebase real-time database back-end with user authentication and device passwords

B. Hardware and Schematics

The device directly plugs into the vehicle's OBD-II port. As such, the device will have an OBD-II connector that shall be soldered on to the product PCB; schematic for this connection is seen in Figure 4.

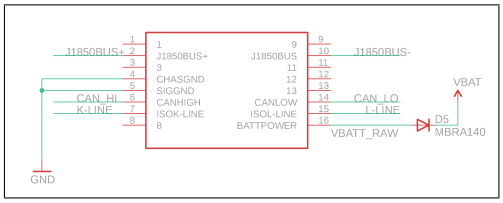

Fig. 4. OBD-II connection

1) Power: The device will be powered through the OBD-II port, which gives 12V-14V, at a minimum amperage of 4A. Pin 16 of the OBD-II connector offers the ability to access the vehicles battery. Battery voltage (VBAT) is stepped down to 5V and 3.3V, as seen in Figure 5, through the use of linear voltage regulators, to supply power to all the components on the device. As the OBD-II port can supply upwards of 4A, the design implements an LM317 voltage regulator to protect from over-current issues.

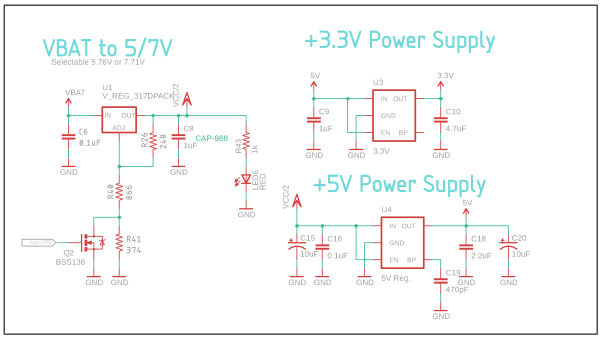

Fig. 5. Power conversion circuitry to convert vehicle battery input voltage to 5V and 3.3V as required by the device

2) OBD-II to UART Conversion: The STN1110 OBD-II-to-UART interpreter is a common IC used in many OBD-II applications. For instance, OBD-II dongles that use this IC are used by mechanics to troubleshoot issues within a vehicle, and by hobbyists to collect vehicle data. This section of the design was inspired mainly from the Sparkfun OBD-II to UART breakout board.

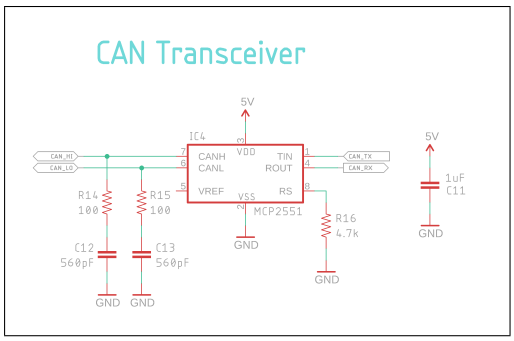

Fig. 6. MCP2515 CAN transceiver IC used to convert CAN Hi/Lo to CAN Rx/Tx

The input CAN high/low signals from OBD-II port are converted to CAN Rx/Tx through a MCP2515 CAN transceiver IC (Figure 6), and the CAN Rx/Tx then gets converted to UART Rx/Tx through an STN1110 multi-protocol OBD interpreter IC (Figure 7).

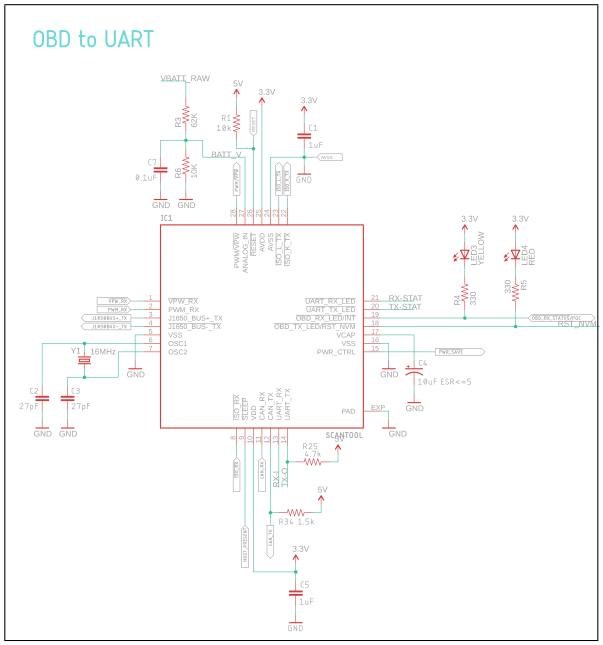

Fig. 7. STN1110 IC and peripheral circuitry to translate between UART and CAN Hi/Lo messages from the vehicle

This allows for two-way communication between the Raspberry Pi (RPi) and the vehicle the device is connected to. The device requires logic level conversion, as explained in Figure 8.

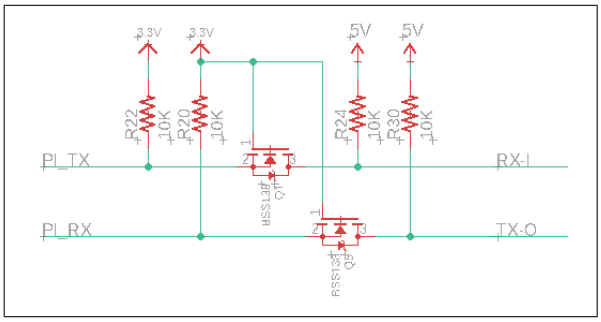

Fig. 8. BSS138 mosFETs are used to convert between 3.3V and 5V logic levels for UART communication between the Raspberry Pi and the STN1110 chip

C. Detection

The detection module utilizes an ADXL345 3-axis digital MEMS accelerometer set to the ± 8g range. The ADXL345 shall sample 3-axis acceleration values at 100Hz, and transmit that information to the Raspberry Pi via I2C.

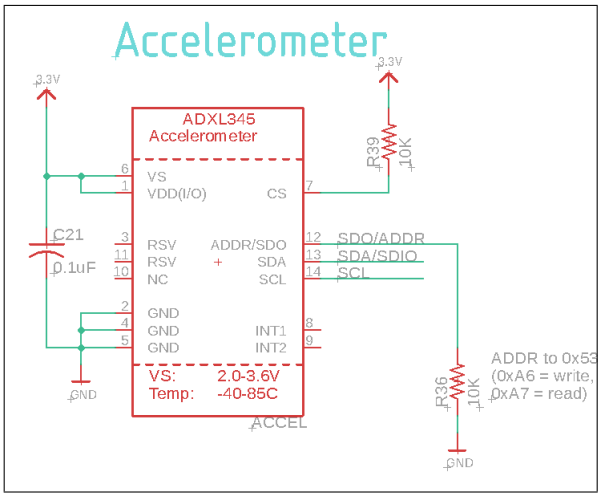

Fig. 9. ADXL345 digital MEMS accelerometer, I2C bus address set to 0x53.

The RPi shall process this information and when the acceleration value is **greater than 3g within a 0.5 second window** (in the x- or y-direction), the device shall flag that as a collision and begin the immobilizing process, as described in later sections. Change in acceleration was the chosen method for detection, as significant shocks to the vehicle would cause a spike in acceleration. Additionally, typical airbag systems utilize accelerometers and deploy after a 5g acceleration pulse is detected. The threshold of 3g was determined using the testing methods discussed in Section VI-E.

D. Immobilization

To immobilize the vehicle after a collision is detected, the device shall query the vehicle speed and determine when it is safe to start applying the brakes. In this design, a speed of under 5km/h was chosen as a safe range to initialize immobilization. This step prevents application of brakes in

unsafe driving conditions, such as icy patches, which could lead to the vehicle losing control.

Once the device determines that the vehicle is safe to immobilize, the device shall send CAN packets via the OBD-II port of the vehicle and into the CAN Network, through circuitry as described in Figure 7. The sent CAN packets will be such that one of the following actions take place and immobilizes the vehicle:

1) Actuation of brakes (preferred)
2) Transmission switched to parked gear
3) Killing the ignition
4) Shut off engine, by cutting off fuel supply
5) Denial-of-Service attack on the CAN network

The preferred method for immobilization is to actuate the brakes to cause the vehicle to remain stationary, while keeping the engine running to ensure passenger safety in harsh climates.

The messages that actuate the brakes of different vehicles are proprietary. To overcome this, product development will require sniffing packets on different vehicles and building a library of valid commands, on a per vehicle basis. The packets sent will be in the standard CAN format, and the required packets for certain vehicles can be found from research conducted on vehicle security. For example, the following is the packet responsible for actuating the brakes and acceleration during cruise control in a Toyota Prius [6]:

```
IDH: 02, IDL: 83, Len: 07,
Data: CN 00 S1 S2 ST 00 CS

CN = Counter that iterates from 00-80
S1 = Speed value one
S2 = Speed value two
ST = The current state of the car
00 = Normal
24 = Slight adjustments to speed
84 = Greater adjustments to speed
8C = Forcible adjustments to speed
CS = Checksum
```

The vehicle can be stopped by sending the following packet continuously:

```
IDH: 02, IDL: 83, Len: 07,
Data: 61 00 E0 BE 8C 00 17
```

E. Disabling

The immobilization device shall be disabled by commanding the processor to stop sending the immobilization packets. This shall restore the vehicle to its unaltered state. The device is disabled through an Android app installed on an authorized personnel's smartphone. The steps involved in using the app to disable the device is described in Figure 10.

The Android app is connected to a real-time database (Google Firebase) which contains authentication information (username/password), as well as unique unlock codes tied to each immobilization device's Bluetooth MAC address. The Bluetooth MAC address is used as the primary key within Firebase, because it is built into the physical layer of the device and thus is unique to each device and cannot be modified.

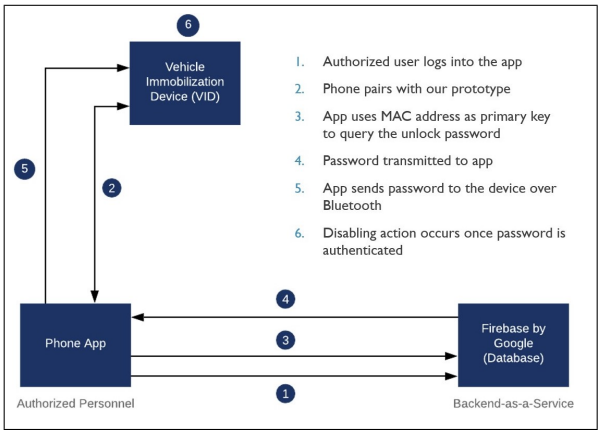

Fig. 10. Process Flow Diagram for Disabling Module

F. Software Implementation

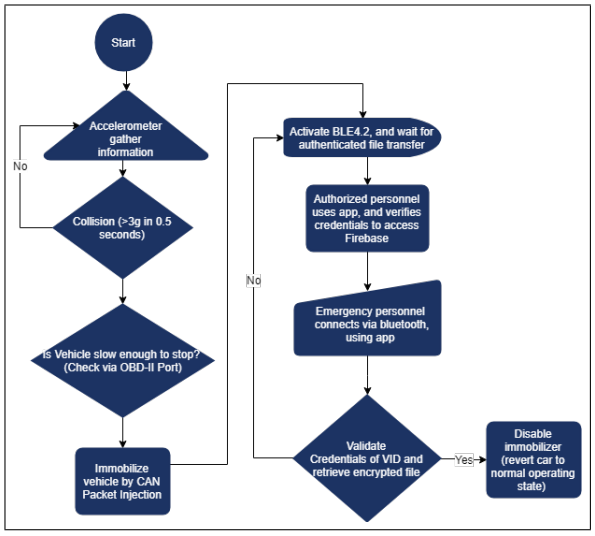

Fig. 11. System flowchart for this device.

1) Raspberry Pi: The main processor in the design is the Raspberry Pi Zero W running the Raspbian operating system. From a software perspective, there are various tasks that need to be performed to incorporate a fully working solution. The immobilization device software incorporates three main separate programs, with scripts nested within it:

- *Accident Detection Program* - Python program that queries acceleration information and determines if a collision has occurred. This program utilizes the ADXL345 library [7] (should be noted that this library had bugs that required modifications that were made during development), and a standard RPi I2C library named smbus.
- *CANBUS Controller* - a C++ based program that handles all communications to the vehicle via the OBD-II to UART circuitry.

- *Disabling Program* - Python program that enables Bluetooth functionality, waits for password, and verifies reception of correct key. This script utilizes terminal commands available through the standard Bluetooth stack for Raspbian [8].

These three programs are implemented to run in parallel as one Linux-based *systemd* service that starts automatically at boot of the RPi. The overall architecture of these programs is based upon a simple Moore Machine, in which the output values are only determined by its current state. As such, the programs need to communicate with each other to indicate any changes to the state through Inter-Process Communication (IPC). The implementation of IPC involves the use of the ZeroMQ (ZMQ) networking libraries, to implement an asynchronous publisher-subscriber (PUB-SUB) based model. TCP/IP Sockets (5556,5557,5558) are used for messages on the local-host through usage of ZMQ library. Any message published is received immediately by all subscribers to the topic, allowing the implementation to be event-driven. This way, state transitions in the separate programs are determined by the current message in the queue. Table II describes what socket each program publishes on and subscribes to, in order to ensure state transitions are handled properly.

TABLE II
PUBLISHER/SUBSCRIBER MODEL

Program	**Publish on**	**Subscribed to**
Accident Detection Program	5556	5558
CANBUS Controller	5557	5556 + 5558
Disabling Program	5558	5557

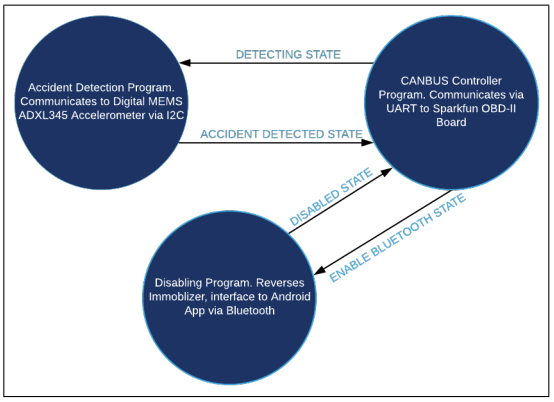

Fig. 12. Software state diagram

The device initializes with “detecting” in the queue. After a collision, *Accident Detection Program* publishes “detected” to the *CANBUS Controller* program, which upon receiving that message shall start immobilization. After successful immobilization, “enable bluetooth” is published to the *Disabling Program* to start associated services and wait for unlock file. Once unlocked, “disabled” message is sent to the *CANBUS Controller* program and immobilization is reversed, placing the device back to “detecting” state. When *Accident Detection Program* sees that message, the script starts to query information again, to start detecting for accidents.

2) Android App: The Android app was programmed using Java and has only two screens for ease of use. The first screen allows the user to log-in using their authorized username and password, which will permit access to the second screen of the app which performs the main function of the disabling module:

- Enable Bluetooth, set the phone as discoverable, discover nearby Bluetooth devices and pair with the immobilizer device
- Query the database using the MAC address to retrieve the correct unlock password associated with the device
- Send the unlock password as a plaintext file to the immobilizer device through Bluetooth file transfer.

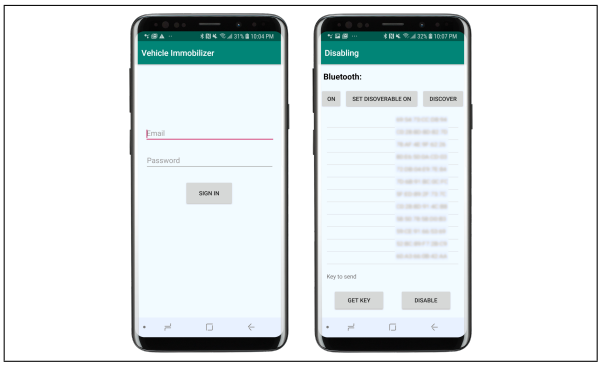

Fig. 13. Android App for disabling the vehicle immobilizer device

IV. PRODUCT CASING

The device is housed in a carbon-fibre casing, a preliminary rendering of which can be seen in Figure 14. The RPi will be mounted on top of the product board using female headers.

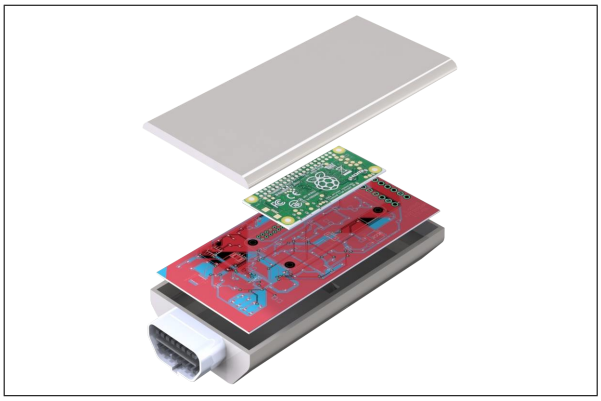

Fig. 14. Exploded SolidWorks view of the product design, dimensions of 132 mm x 65 mm x 25 mm

1) Locking mechanism: To ensure that the product is tamper-proof, a locking mechanism will be present on the front of the device. This tension-based spring locking mechanism, together with retracting fins on the side of the port, will prevent the removal of the product. To remove the device a special

tool is inserted into the side of the product to bypass the aforementioned locking mechanism. The tool will only be available to authorized personnel.

2) Status indication: The state of the device shall be indicated to the end-user through LEDs mounted on the back of the dongle. Each LED serves a specific purpose:

1) Device receiving power
2) All software program are operational and device is actively checking for accidents
3) Collision detected and immobilization packets being sent

When the device is disabled, LED 3 turns off and LED 2 comes back on. If the device is in fault condition due to software error, both LED 2 and 3 are turned on.

V. Cost of Manufacturing

Table III breaks down the cost of manufacturing each device by taking into account all the components that will be used in the device, as described in Figure 14; the total cost is under CAD $100 per device.

TABLE III
Cost of Manufacturing

Item	Cost
Electronic components (digikey.ca)	$75
Board fabrication (JLCPCB.com)	$10
Material cost for housing	$5

VI. Testing and Results

Testing on an actual vehicle would have been the ideal way to test the device's functionality. However, due to the hazards associated with using a vehicle, an alternative approach was used. A proof-of-concept device was built with all the modules discussed in III, and a custom ECU simulator was constructed to test the device instead of an actual vehicle.

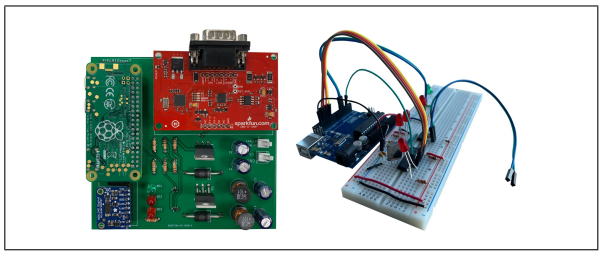

Fig. 15. Proof of concept device (left) and the custom ECU Simulator (right).

A. Proof-of-concept design

The proof-of-concept (POC) device (Figure 15) was developed by building around a Sparkfun OBD-II to UART conversion breakout board which utilizes the STN1110 chip. The Adafruit ADXL345 breakout board is used for the detection module. The power module on the POC incorporates the same design as described in the design section with through-hole components. The main processor for the POC device is a Raspberry Pi Zero with the software described in Section III-F1.

B. Custom ECU Simulator

This testing scheme eliminates the need to procure a real vehicle, and mitigates risks associated with testing in the real world.

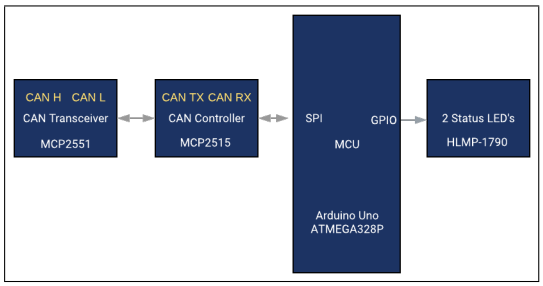

Fig. 16. Hardware block diagram of the custom ECU simulator

The simulator's hardware block diagram, Figure 16, shows the main components used to build it. The Arduino was programmed to simulate an ECU by incorporating the following functions:

- Respond to standard OBD-II parameter queries, such as returning a randomly generated vehicle speed
- Indicate normal vehicle operation through a green LED
- Perform an action (turn on a red LED) when receiving specific CAN packets to demonstrate vehicle immobilization

C. Verification

After both the POC device and the ECU simulator were developed, the CAN HI/LO from the OBD-II to UART board of the POC device was connected to the CAN HI/LO of the ECU simulator to establish communication between the two. The following sequence was tested:

1) After power on, the simulator only powered the green LED as expected
2) An accident was simulated by exposing the POC device to a significant shock
   - *Note*: the threshold for detection was lowered at this stage to a value that was easier to obtain by hand in a lab environment
3) The POC detects the accident and starts sending immobilization CAN packets
4) The simulator receives the known immobilization packet and turned off the green LED, while the red LED turned on to signify immobilization
5) The Android app was then used to send the correct unlock key to the POC device
6) The red LED on the simulator went off and the green LED came back on, which demonstrates the vehicle returned to normal state

The above sequence of testing was performed multiple times, and the system performed successfully on each iteration which demonstrates the robustness of the design. During these tests, the CAN network was connected to an oscilloscope to verify the sending and receiving of the data. The captured

packets, example shown in Figure 17, were in the CAN format using the 11-bit format using bit-stuffing. This was expected from our design, and this successfully verifies that the POC and simulator communicate using the CAN protocol.

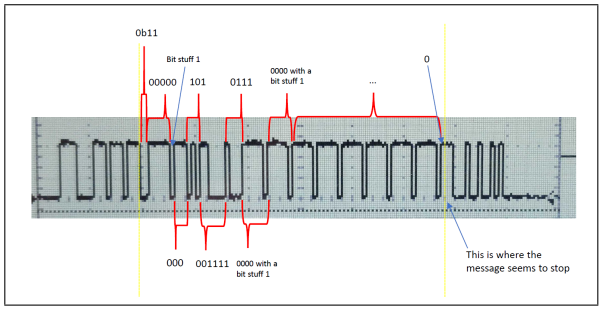

Fig. 17. Sample CAN packet (0xC0 29 EE 00 00 00 00 00) captured from the oscilloscope during testing

D. Power Consumption Analysis

When testing the POC device, the power consumption of each module was noted, along with the power consumption of the overall product. The maximum rated power consumption is shown in Table IV. The POC device was tested using a standard lab variable DC power supply set to 12V. The device drew 220mA at most, which was during startup and settled at around 180mA during normal operation. As the OBD-II port of a standard vehicle outputs a minimum standard of 4A, it would be an adequate power source for the device. With a typical vehicle battery's at 45 amp-hours capacity, this device may drain a vehicle's battery if the vehicle is left stranded for approximately two weeks. However, if the vehicle is driven regularly there should be no issues. To address this though, the design allows for optimizing the power consumption through software as described in section VII-B

TABLE IV
POWER CONSUMPTION

Component	Voltage Level (V)	Rated Current(mA)	Power at Rated Current (W)
OBD-II to UART board	12V	80	0.96
Raspberry Pi Zero W	5V	1000	5
Accelerometer Board	3.3V	0.14	0.000462

E. Testing acceleration spikes

The accelerometer was tested by monitoring its output while subjecting the device through different movements. First, the accelerometer was moved around in space to verify that it operated and was calibrated properly. Afterwards, data was acquired and the threshold for classifying a collision was developed, using test runs during controlled driving conditions. This was done to filter out typical shocks and jerks that occur during normal driving, as well as situations that could potentially be classified as a collision by the device. The following false-positive cases were examined:

1) Passing over potholes, a maximum of 2g change in acceleration was noted in the z-direction
2) Passing over a speed bump, also resulted in acceleration only in the z-direction
3) Hard braking, as seen in Figure 18, was conducted in a safe environment and a maximum of 1.05g of braking force was seen in the x-direction (facing the front of the vehicle)

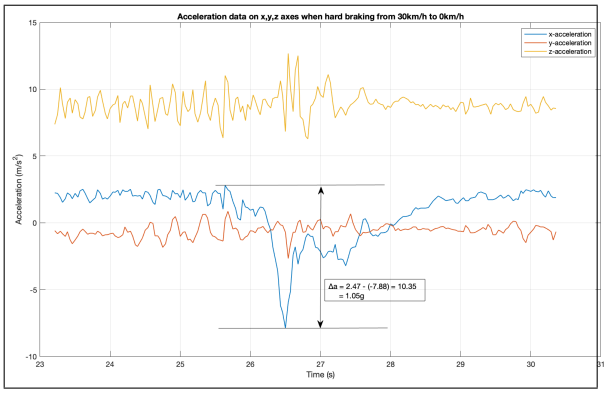

Fig. 18. Acceleration spike after hard braking

Due to most of the false positives being in the z-direction, acceleration data in this direction is not used by the POC device to classify collisions; only x- and y-axes are used. Since a maximum of 1.05g was achieved while hard braking, a safety factor of 2g was added, leading to a threshold of 3g, to be used by the device to declare an accident.

VII. Future Considerations

Various features can be developed to optimize, and to enhance the performance of product. These functions are not critical to the proof-of-concept of the methodology, however they improve usability. Few of the immediate goals have been identified in this section.

A. CAN Packet Sniffing

A local library containing manufacturer specific immobilization packets can be constructed and stored locally within the device's firmware/software, in the development stage. Once installed on vehicle, the device can identify the vehicle by querying the Vehicle Identification Number (VIN) number using the corresponding OBD-II command to determine the library entry that will be accessed for immobilization. The RPi has internet capability, thus Over-the-Air firmware updates can update the library for new vehicle models and other changes, as deemed necessary. Development of library can be achieved via monitoring CAN traffic of vehicles while manually applying the brakes to find braking packets in that particular vehicle. The library of valid packets can be built by obtaining this information from multiple vehicles and testing those packets on the vehicle.

B. Low power mode

A low-power mode when the device does not require the full power is to be implemented, to decrease overall energy consumption of device. The device will be drawing power from a vehicles battery even when the engine is off, and can fully drain the battery in $\tilde{2}$ weeks. Therefore, when the vehicle is stationary (and no collision has been detected), device could be put into a low-power state to consume limited power by checking:

- If the vehicle is stationary - through the 'inactivity' interrupt flag from the accelerometer
- If the engine is off - Via OBD-II PID Query

C. Improved detection

Detection of various types of collisions will enhance the performance and effectiveness of the device. The accelerometer data can be gathered in all three axes of motion. The direction in which the acceleration spike occurs, allows the ability to detect various types of collisions instead of only front-end collisions. Furthermore, severity of collision can be determined by analyzing the size of the spike and using fuzzy logic to characterize the severity.

As acceleration threshold gets lowered, noise in the system may become a large factor, and additional sensors or redundant data may need to be gathered to accurately detect collision. To perform this, accessing additional data sources from the CAN bus can be performed through minor software/firmware modifications. These additional data sources can be processed in conjunction with the data retrieved from the accelerometer to improve the accident detection algorithm.

D. Improved Security

To prevent intruders, the device can be programmed to be more secure and tamper-proof. For this purpose, dynamic encryption keys can be used. RSA authentication is one of the widely used method for this purpose. Per RSA guidelines, each device will have a private-public key, which can only be decoded by a unique mathematical function. The public key will reside in the back-end database server, while the private key will reside in the device. Additionally, multi-factor authentication can be implemented to further improve security. Multi-Factor authentication involves multiple authentication measures that a user has to undergo to successfully complete the authentication process.

E. Location Information

After a severe collision, it is crucial for authorities to promptly arrive at the scene. To aid the authorities, a GPS/GSM module can be integrated within the device to notify emergency response personnel of the location of the accident, allowing for a swift response to such a severe collision.

VIII. Patent Analysis

To ensure that this device did not infringe on patented products in the market, a patent analysis was conducted. The following patents were analyzed:

1) vehicle information acquisition device: obtains vehicle information using the CAN network, however the device is based around obtaining information from the CAN network and does not seek to immobilize the vehicle.
2) CAN based vehicle immobilizer [9]: this device is similar to our design in that, it uses the CAN bus to immobilize the vehicle, however it is done by shorting the CAN high and low lines together to prevent communication, rather than targeting specific ECUs to perform actions that result in immobilization
3) diagnostic port intoxication vehicle immobilization [10]: seeks to immobilize the vehicle if a breathalyzer detects alcohol in the drivers system.

Based on preliminary patent analysis, there are no devices or patents that the design of the VID infringes.

IX. Conclusion

Overall, the design of the Vehicle Immobilization Device has been proven and tested to work through the prototyping stage. The following objectives were met: accident detection, immobilization of vehicle, and reversal of the immobilization. Along with this, the design meets criteria such as scalability, reliability, longevity, tamper-proof, ease-of-installation, and weather considerations. Based on the proof-of-concept design, detailed design on the final commercialization product has been started, as indicated above.

ACKNOWLEDGMENTS

We would like to thank the following people for making the project a success: Loren Wyard-Scott, Dr. Michael Lipsett, Peng Zhuang, Alan Lim, Ben Flanders, Jesse Tham, Simarjeet Dhanoa, Victor Zhao, Tyler Trinh, Eric Der, Graham Hornig, and Bianca Angotti.

References

[1] A. T. O. of Traffic Safety, "Alberta traffic collision statistics 2015," Tech. Rep., 2016, [Accessed 21-Jan-2019]. [Online]. Available: https://www.transportation.alberta.ca/Content/docType47/Production/AR2015.pdf
[2] S. Corrigan, "Introduction to the controller area network," Texas Instruments, Standard, 2016.
[3] Dhananjayan. (2018) Working with automotive can protocol. [Accessed Mar 21 2019]. [Online]. Available: http://www.embien.com/blog/working-automotive-can-protocol/
[4] O.-I. H. Page. (2011) Obd-ii - on-board diagnostic's system - does my car have obd-ii? the connector and communications. [Accessed Mar 28 2019]. [Online]. Available: http://www.obdii.com/connector.html#dates
[5] C. Electronics, "Obd2 explained - a simple introduction," 2019. [Online]. Available: https://www.csselectronics.com/screen/page/simple-intro-obd2-explained
[6] D. C. Miller and C. Valasek, "Can message injection," Tech. Rep., 2016. [Online]. Available: http://illmatics.com/can%20message%20injection.pdf
[7] T. DiCola and Kattni, "Adafruit python adxl345 library," https://github.com/adafruit/Adafruit_Python_ADXL345, d897bcaa16b5fe365463ae0c2791d0c155becd05, 2016.
[8] A. O. E. al., "bluez-tools," https://github.com/khvzak/bluez-tools,
[9] S. A. Tarnutzer and T. Prohaszka, "Can based vehicle immobilizer," U.S. Patent US8 918 251B2, Dec. 23, 2014.
[10] T. J. M. Douglas Edward Devries, Kerry L. Keller, "Diagnostic port intoxication vehicle immobilization," Canada Patent CA2 938 407C, Aug. 10, 2018.