Automatic Reverse Engineering of CAN Bus Data Using Machine Learning Techniques

Thomas Huybrechts[1(✉)], Yon Vanommeslaeghe[1], Dries Blontrock[1], Gregory Van Barel[2], and Peter Hellinckx[1]

[1] Imec, IDLab, Faculty of Applied Engineering, University of Antwerp, Antwerp, Belgium
{thomas.huybrechts,peter.hellinckx}@uantwerpen.be,
{yon.vanommeslaeghe,dries.blontrock}@student.uantwerpen.be

[2] Op3Mech, Faculty of Applied Engineering, University of Antwerp, Antwerp, Belgium
gregory.vanbarel@uantwerpen.be

Abstract. The CAN (Controller Area Network) bus connects different *Electronic Control Units* (ECU) inside a vehicle. Valuable information about the state of the vehicle is present on this bus and is useful to track driver behaviour, the health of the vehicle, etc. However, the configuration of this bus is not publicly disclosed by the car manufacturers. Therefore, reverse engineering techniques need to be applied. Nevertheless, performing these techniques manually is cumbersome and time consuming. In this paper, we propose an automation of the analysis steps of reverse engineering in order to improve and facilitate the process. Two approaches of automation are discussed, namely an arithmetic approach and machine learning using classification. In conclusion, we show that reverse engineering of CAN bus traffic is at least partially possible by applying machine learning techniques and the performance of the classifiers increases when adding additional features to the analysis.

1 Introduction

Modern cars contain complex electronic and mechatronic systems that monitor and control various aspects of the vehicle. These systems work together to increase the safety and comfort of the driver. The different systems communicate over one or multiple CAN (Controller Area Network) buses. The CAN bus is common in all modern car. Luxury cars have up to 100 *Electronic Control Units* (ECU) and more then 1 Km of cables connecting them all. Figure 1 shows a generic CAN bus configuration. These networks are usually divided in different subnetworks according to their function and specific needs, such as high-speed CAN (CAN-C) for the motor management and low-speed CAN (CAN-B) for climate control. As a result, a major part of information about the inner workings of a vehicle can be learned by examining CAN bus traffic. This information is useful to track driver behaviour, the health of the car, etc.

© Springer International Publishing AG 2018
F. Xhafa et al. (eds.), *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, Lecture Notes on Data Engineering and Communications Technologies 13,
https://doi.org/10.1007/978-3-319-69835-9_71

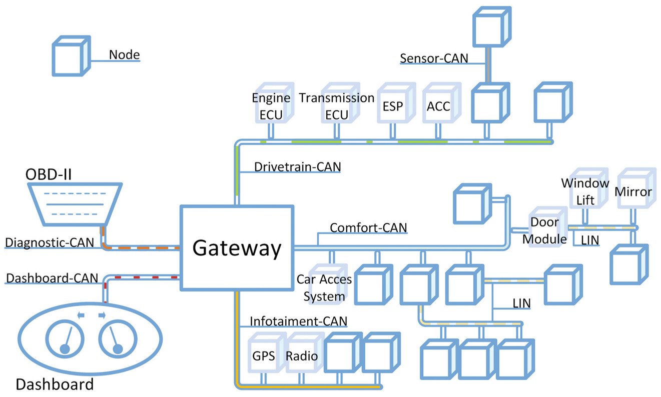

Fig. 1. Generic model of a car CAN bus. Multiple subnetworks with different functionality are interconnected through gateways.

However, the problem is that although the CAN protocol is standardised, the actual implementation differs for every manufacturer and even differs for every model. Automotive manufacturers are free to choose which ID represents which data, how the data is represented, etc. This configuration is not publicly disclosed. This means that CAN bus traffic for every car has to be analysed and reverse engineered in order to obtain useful information. Furthermore, reverse engineering rises legal and ethical questions [1].

Current reverse engineering techniques are relatively straightforward, but time consuming. They rely on trial and error and require knowledge of the inner workings of the vehicle's different subsystems. The time required to reverse engineer a vehicle increases dramatically as the vehicle becomes more complex. Therefore, we applied two different techniques, i.e. arithmetic and machine learning approach, to automate this process, reducing the analysis time and effort while acquiring sound results.

In this paper, we will first discuss state of the art methodologies used to reverse engineer a CAN bus. Second, we present two different approaches to automate the reverse engineering process. Final, we discuss our results and performances of the implemented automation techniques.

2 Controller Area Network

The CAN protocol is a standard developed by Robert Bosch GmbH to create an efficient and reliable communication bus between nodes in real-time applications. Every message transmitted on the CAN bus carries a specific identifier. This identifier does not directly indicate the sending or receiving node, but instead identifies the content of the message.

The CAN standard specifies the general structure of a message, but does not define other aspects, such as byte order, signedness, amount of signals in one message, byte alignment, scale, etc. This approach provides great flexibility, allows manufacturers to efficiently use the CAN bus and basic ‘security’ through obscurity. However, it also significantly increases the complexity of reverse engineering the network.

3 Reverse Engineering

Reverse engineering the traffic on the CAN bus is a relatively straightforward process, but is fairly time consuming. Current techniques rely on manual analysis of the traffic and require knowledge of the inner workings of the vehicle and its different subsystems.

Hermans et al. [4] extended on a technique outlined by Eisenbarth et al. [2] for locating features in source code and applied it to CAN data frames. The proposed reverse engineering process consists of five steps as shown in Fig. 2. This five step methodology seems to be a good summary of reverse engineering techniques for CAN bus data as others often follow these steps, although not explicitly.

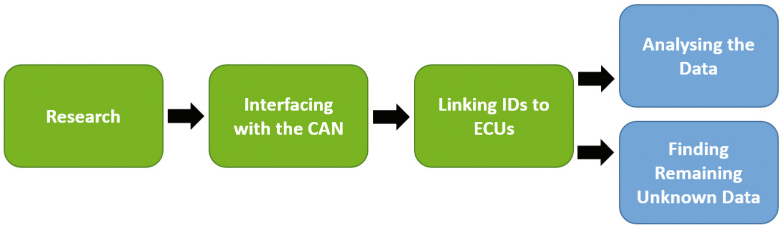

Fig. 2. Flow chart of the five step reverse engineering process.

Research. The first step is investigating the car’s construction and which functionality it has, e.g. ABS, airbags, etc. In addition, it is recommended to look up on which network these functions are located and how these networks interconnect in order to know which data to expect, i.e. the wiring on the electrical schemes.

Interfacing with CAN Bus. The second step is interfacing with the CAN bus. This is the stage where one collects as much data as possible. All the messages will be stored for further analysis. Interfacing with the CAN bus is achievable through either the OBD-I, II plug, if the car supports this functionality, or (in)direct physical contact with the network.

Linking Identifiers to ECUs. The third step is linking the found IDs to their corresponding ECUs. Combined with the previous step, a first indication is provided as to which signals are contained in each message.

For this step, two major methods are presented in literature. A first method is to unplug each control unit one by one and to note which IDs disappear from the network, as presented in [4]. However, Rokicki and Szczuwoski [8] noted that identifying a node by disconnecting it may also change the behaviour of other nodes in the network which may result in an incomplete picture of the network. Therefore, they introduced their own solution, which is to insert their own two nodes between the target node and the rest of the networks. These two nodes act as gateways and allow them to observe the direction of the data flow. However, Freiberger et al. [3] concluded that the approach of [8] is less error-prone, but more time consuming compared to [4] because of the rewiring needed when inserting a gateway between a node and the CAN network.

Analysing the Data. The fourth step is finding the *essential* data in the recorded data sets. During this step, associations need to be found between the data points and features, e.g. a correlation can probably be revealed when the speed is monitored in relation to time and so, linking a certain ID to the messages that represent the speed. To find the essential data, the researcher has to complete the tedious task of plotting all data and finding matches in the acquired data sets.

When acquiring CAN data, we need to consider what test cases to perform. Depending on a clever selection of cases, this step becomes easier to accomplish. A simple example would be to turn on the car, but leaving the engine off while moving the throttle pedal in order to determine the throttle position.

Finding Remaining Unknown Data. The final step is analysing the remaining *unknown* data. The possible combinations of IDs with the remainder of data can be reduced even further when more IDs get linked in the process. The fourth and fifth step are performed in parallel, as indicated in Fig. 2.

4 Automating the Process

Performing the reverse engineering process manually as described in Sect. 3 is exceedingly time consuming [3]. The major part of the effort is needed during the last three steps of the process as shown in Fig. 2. The goal is to automate these steps as much as possible so that we are able to reduce the manual labour and cut down the high data analysis time. In this paper, our main focus is on reducing the data analysis part of the process. Therefore, we applied different data analysis methodologies and evaluated the effectiveness of these approaches.

Arithmetic Approach. A first approach that we investigated is to use an arithmetic methodology with the *Root Mean Square* (RMS). This technique allows us to compare two signals and calculates the deviation between them. As a reference signal, the data obtained from the *On-Board Diagnostics* (OBD) connector is used. With OBD, we are able to request standardised real-time data if the car supports the desired *Parameter IDs* (PID), such as engine RPM, vehicle speed, etc. The algorithm will compare the OBD signal with every byte and selected combinations of bytes in the CAN messages. The RMS calculates the deviation of every data point. The results with the lowest average deviation has a higher probability to represent the wanted signal. In case that OBD does not provide the wanted signal, we will use simulation to acquire the necessary signals. For these simulations, we use the AMESim tool to run a simulated car model. This simulation provides us a realistic representation of the system values. It is important to note that the simulated signals do not necessarily have the same formatting as data on the CAN bus, e.g. different offset, scale, etc. By applying a normalisation on the data, better results are obtained.

Machine Learning. In essence, the classification of CAN data is a time series classification problem. Many classification techniques exist, but most of them are designed for situations where there are multiple input features and an associated class. The difference with time series data is that these input features are not independent from each other, i.e. their order matters. However, it is still possible to use regular classification techniques by calculating different features based on the series and using those for classification.

In recent years, neural networks have become popular for machine learning tasks. The development of *Convolutional Neural Networks* (CNN) caused a massive increase in accuracy in tasks, such as image classification [6]. With some modifications, these networks can also be used for time series classification [10].

Neural networks such as CNNs are stateless, they have no memory of previous inputs. This is a disadvantage for time series classification, as the previous states of a signal provide valuable information. Therefore, a so-called *Recurrent Neural Network* (RNN) has an internal loop that serves as an internal memory [7]. This allows the network to take context into account. An example of RNN is the *Long Short-Term Memory* (LSTM) network, these types of neural networks perform particularly well at pattern recognition tasks and have revolutionised speech recognition [5,9].

In order to train a sound and useful neural network, lots of data is needed. In the following sections, a brief discussion is made on how the data sets were acquired and processed as data plays a major role in machine learning.

Data Acquisition. For our research, we performed experiments on two different cars, namely a 2007 Ford Focus and a 2012 Hyundai Veloster 1.6 GDI. In order to acquire the CAN data, we made a connection through the OBD connector with a Kvaser Leaf SemiPro. The CAN traffic is recorded by a self-developed tool which allows us to incorporate global timestamps for synchronisation. Besides recording CAN traffic, we collected GPS data from the Racelogic VBOX 3i Dual Antenna to enrich our data with extra features, such as latitude and longitude, speed, heading, etc.

The data used is collected by simultaneously recording CAN traffic and GPS data while driving. The data is then processed and filtered to build a training dataset for the different classifiers. In addition, each classifier is trained once with and without GPS data in order to compare their performance. The idea behind using positioning and orientation data as extra features is that it might provide extra knowledge of the vehicle state that could be used to more accurately classify each signal.

Data processing. After data acquisition, we extract all signals of interest from the recorded CAN traffic. In this experiment, we selected a couple of signals as features to train the classifiers, i.e. throttle position, brake pressure, engine RPM, vehicle speed and steering angle. Other signals such as counters and intermittent signals are used as well which are classified as '*other*'.

Messages on the CAN bus are transmitted at different rates up to 1 Mbit/s. Additionally, these rates are not constant due to message preemption. The data recorded on the VBOX however is at a constant 100 Hz. It is important to have a constant sampling rate as synchronisation between data points is crucial. Therefore, the CAN and GPS data is resampled and interpolated at a rate of 10 Hz, as it was the lowest among the signals of interest. Resampling the data at the lowest rate is more convenient, since upsampling might introduce extra features. These features could cause false results as the classifiers might learn to use them to classify signals.

Signals on the CAN bus may be represented in different formats, e.g. offset, scale, etc. In order to create generic working classifiers, all data sets have to be normalised. Otherwise, classifiers might learn to classify based on the scale or offset of signals. As a result, a classifier is obtained which will only work on vehicles with the same data formatting. Therefore, we scale each signal in the interval [0, 1].

A similar operation is performed on the speed from the GPS data. Although the scale will be the same for every vehicle, some of the classifiers used are sensitive to scaling. During testing, it became apparent that momentary loss of the GPS signal causes false speed readings that are often much higher than the actual speed. To avoid problems due to these false readings, the speed data is normalised only with readings where the number of satellites is above a certain threshold.

Sliding window. Before feeding the data to the classifiers, a window technique is applied to the CAN data. This allows us to split the input/test sets into multiple subsets of a fixed length. A first advantage of applying a sliding window is that it significantly increases the amount of samples in the dataset. Second, the CNN and LSTM require a fixed data input size which the sliding window provides.

Identifying Signals. Each classifier is trained on signals that are manually extracted from recorded data. However, this is not the actual aim of automation as we want to be able to determine the meaning of unseen CAN data without manual effort. In order to find signals in the CAN traffic, an adjustable window scans across the CAN messages. For example, first classifying on byte level, then every subsequent two bytes, etc. However, CAN messages may contain multiple signals of any length, with any byte order and signedness. Just trying every combination will become computationally expensive very fast. A better approach is to gradually identify signals by first evaluating at byte level and then, when the confidence of the classification is above a threshold, narrowing down the search interval. Therefore, identifying the exact location of the signal. The result with the majority score determines the final class. For our research, we only considered sequences with one or two bytes.

5 Results

All results in this paper were acquired from a Hyundai Veloster (2012) and a Ford Focus (2007). In the following sections, we discuss the results for each methodology described in Sect. 4.

Arithmetic Approach. In a first step, we performed multiple test drives while recording the CAN bus and diagnostic information from the OBD connector. The requested data from the OBD contains the throttle position, vehicle speed and engine RPM. The ten best results from the RMS method are shown in Table 1A. At this point, a visual inspection is required to verify the results. In Fig. 3, we plotted the vehicle speed with the four CAN signals which have the smallest RMS deviation. The first three results, e.g. A0-4, 18F-3 and 316-6, are a close fit compared to the OBD speed with PID 7E8-3 and resemble the vehicle speed. The latter result however, represents the first byte of the RPM. We know this from the data we acquired from manually reverse engineering the car.

To verify our results, we also applied the method using simulation data instead of OBD data. The results are shown in Table 1B. The three correct IDs we acquired from the OBD tests are also present in the simulation results. Nevertheless, they are less accurate. These deviations in the simulation derived from measurements errors of the external sensors used during the test drives.

By applying this methodology, we already matched 24 of the 208 bytes to the corresponding data on the Hyundai Veloster. However, these results are heavily influenced by the information recorded on the CAN bus and OBD standard.

Table 1. Vehicle speed results for RMS method with OBD and simulation

(A) OBD			(B) Simulation		
ID	Byte	Deviation	ID	Byte	Deviation
A0	**4**	**0.003065**	2A0	2-5	0.0
18F	**3**	**0.004920**	**18F**	**3**	**0.006131**
316	**6**	**0.004920**	**316**	**6**	**0.006134**
80	3	0.052117	1F1	7	0.006272
316	3	0.052117	1F1	4	0.006716
1F1	4	0.066539	**A0**	**4**	**0.011192**
2A0	4	0.077394	80	5	0.021227
329	3	0.080275	316	5	0.021227
329	4	0.080775	545	3	0.022280
164	1	0.080838	4B0	7	0.022388

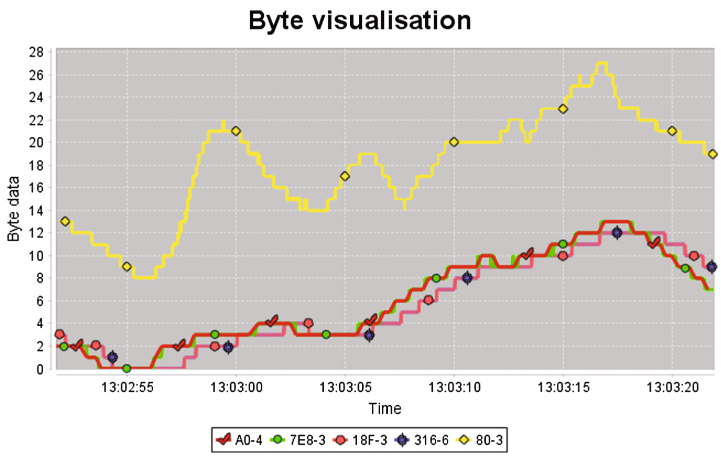

Fig. 3. Signal plot of CAN data with lowest deviations compared to OBD vehicle speed (7E8-3).

In addition, about half of the recorded data contain either counters or constant flags which never changed throughout recording.

Machine Learning. Initially, four versions of every classifier were trained, two pairs with respectively only CAN data and CAN combined with GPS data. Each pair then contains a version where all features are used and one with a greedy algorithm implementation, except for the CNN and LSTM network. The overall performance of the classifiers is shown in Table 2 where the best combination for each classifier are highlighted.

The results indicate that adding GPS data increases the performance of most classifiers. Some classifiers however do not benefit from the extra information. Firstly, the greedy algorithm selects the same features for these classifiers when only CAN data as both CAN and GPS data are provided. Secondly, neural

Table 2. Overall performance of the different classifiers

	CAN	CAN+GPS	CAN greedy	CAN+GPS greedy
Discriminant analysis	64.9	65.6	71.9	**73.0**
K nearest neighbours	63.1	68.6	**76.4**	**76.4**
Logistic regression	68.8	67.3	74.2	**78.1**
Naive bayes	66.5	66.4	**71.9**	**71.9**
Support Vector Machine	64.2	69.8	74.0	**78.2**
CNN	91.0	**93.8**		
LSTM	**90.8**	87.7		

networks perform better than the classic techniques in general when comparing the results. Nevertheless, the benefit of adding GPS data becomes more apparent when we use these neural networks to identify signals in the entire CAN traffic instead of just using test sets as is the case for Table 2.

The results for trained CNNs in Table 3 show the possible matches and confidence scores to discover speed signals in new data. We found that we are able to identify both vehicle speed signals and all four wheel speeds. However, a false positive with a relatively high confidence score was present.

As we use the trained CNN to seek other signals of interest in unseen data, additional observations were made. The position of the *throttle* pedal is represented by both a one and two byte signal which were correctly identified by the CNN. One of the signals was unknown to us before. Additionally, multiple other signals were identified as representing throttle position. However, they seem to symbolise related values, such as throttle body position or other engine parameters. The actual *RPM* signal of the engine was correctly identified. Nevertheless, two other signals were incorrectly identified with similar confidences. The *brake pressure* was identified with a significant confidence. Other '*incorrect*' signals had considerably lower scores. Similar observations were found for the *steering angle* compared to the brake pressure, as the correct signal had a remarkably higher confidence score.

The previous results were obtained from the Ford Focus on which the classifiers were initially trained. To verify the proposed method and its generic properties, we performed the analysis on a '*new*' car, i.e. Hyundai Veloster.

Table 3. Results when looking for speed signals

	Only CAN	CAN + GPS
Possible matches[1]	20	10
Vehicle speed 1[2]	0.65	0.71
Vehicle speed 2[2]	0.60	0.64
Wheel speed FL[2]	0.41	0.48
Wheel speed FR[2]	0.41	0.55
Wheel speed RL[2]	0.16	0.23
Wheel speed RR[2]	0.23	0.42

[1] A possible match means a confidence higher than random chance (1/number of classes).
[2] Confidence score

Two vehicle *speed* signals were identify, however only half of them were discovered. As opposed to the other vehicle, the signals are not byte aligned or include no other signals in the same byte. However, searching for more different signal lengths should solve this problem. The *throttle position* has similar results as the Ford. Yet, two false positives with a high confidence were indicated. This may be due to the resampling operation on these signals which are normally sent at a slower rate of 1 Hz. When attempting to identify the *engine RPM*, the signals with the highest confidence seemed to be related, but did not directly represent it. Although the correct signals were still identified, their confidence score were significant lower than the related signals. The results for the *brake pressure* were poor, as the first signal found was incorrect. The actual signal may represent the vacuum pressure. Even though it is related, it is not at all similar to the training signal. The second signal indicated actually presents steering torque. The false positives may be due to a correlation between the two as a driver normally slows down before turning. For the *steering angle*, there was initially no match. We found that the Hyundai used a signed representation, which is opposed to the offset used on the Ford. When taking this into account, the steering angle was identified correctly without any other matches.

6 Conclusion

The manual process of reverse engineering a CAN network needs immense effort and is time consuming. Interpreting all resulting data requires know-how and time. Automating the analysis offers a considerable overall improvement. In this paper, we suggested two approaches to automate the analysis. The first arithmetic approach is able to match OBD or simulated data to a specific location in the CAN bus traffic. However, this approach still requires visual inspections and its effectiveness greatly depends on the vehicle itself, e.g. data formatting, available data on OBD, etc.

The second approach involves machine learning techniques. The results show the possibilities to, at least partially, identifying signals of interest in the data of CAN bus traffic. Additionally, we have proven to increase the performance of certain classifiers used by including '*other*' data about the state of the vehicle, e.g. GPS speed. However, more experiments on larger datasets and different vehicles need to be performed in order to validate our findings. Furthermore, only signals of one or two bytes were considered in these experiments. In reality, there are many more possibilities. Simply trying ever possible combination will become computationally expensive really quickly. It is therefore useful to look into more efficient techniques to identify the position of signals in CAN data.

Since the promising impact of adding GPS speed as a feature to the classification, future research can be conducted to evaluate the impact of additional data features in the classification process. Therefore, we propose to develop a hardware solution that contains a CAN transceiver with additional sensors, such as GPS and an IMU to collect more synchronised data on different vehicles. Eventually, we want to create a device which will be able to perform the reverse engineering process by itself.

References

1. Currie, R.: Hacking the can bus: basic manipulation of a modern automobile through can bus reverse engineering. Technical report, SANS Institute (2017)
2. Eisenbarth, T., Koschke, R., Simon, D.: Locating features in source code. IEEE Trans. Softw. Eng. **29**(3), 210–224 (2003). doi:10.1109/TSE.2003.1183929
3. Freiberger, S., Albrecht, M., Käufl, J.: Reverse engineering technologies for remanufacturing of automotive systems communicating via can bus. In: Glocalized Solutions for Sustainability in Manufacturing - Proceedings of the 18th CIRP International Conference on Life Cycle Engineering, pp. 90–95 (2011). doi:10.1007/978-3-642-19692-8-16
4. Hermans, T., Denil, J., et al.: Decoding of data on a can powertrain network. In: 16th Annual Symposium on Communication and Vehicular Technology in the Benelux (1), 6 (2009)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997). doi:10.1162/neco.1997.9.8.1735
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012)
7. Mandic, D.P., Chambers, J.A., et al.: Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability. Wiley Online Library (2001)
8. Rokicki, K., Szczurowski, K.: Methods of identification of data transmitted in the in-vehicle can-bus networks. In: 2015 20th International Conference on Methods and Models in Automation and Robotics, MMAR 2015, pp. 946–949 (2015). doi:10.1109/MMAR.2015.7284005
9. Sak, H., Senior, A.W., Beaufays, F.: Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. CoRR **abs/1402.1128** (2014)
10. Zheng, Y., Liu, Q., et al.: Time series classification using multi-channels deep convolutional neural networks. In: WAIM 2014: Web-Age Information Management, pp. 298–310 (2014)