AUTOMATED REVERSE ENGINEERING OF CAN PROTOCOLS

N. Weiß[*], *E. Pozzobon*[*], *J. Mottok*[*], *V. Matoušek*[†]

Abstract: Car manufacturers define proprietary protocols to be used inside their vehicular networks, which are kept an industrial secret, therefore impeding independent researchers from extracting information from these networks. This article describes a statistical and a neural network approach that allows reverse engineering proprietary controller area network (CAN)-protocols assuming they were designed using the data base CAN (DBC) file format. The proposed algorithms are tested with CAN traces taken from a real car. We show that our approaches can correctly reverse engineer CAN messages in an automated manner.

Key words: *controller area networks, statistical analysis, neural networks*

Received: February 7, 2021 **DOI:** 10.14311/NNW.2021.31.015
Revised and accepted: August 30, 2021

1. Introduction

The major communication technology for real-time data exchange in modern cars are CANs [7]. CAN is a message-based communication protocol with an identifier field and eight bytes of payload. During the normal operation of a car, the identifier field of a CAN message is used to specify the type of the transported data. The eight bytes data field contains various real-time values. Every message is broadcast on a shared bus. An electronic control unit (ECU) can filter on the message identifier to evaluate if a message is relevant for its internal functionality. To define the data which is transported over CAN, original equipment manufacturers (OEMs) are using DBC files or similar description formats for their internal development. These DBC files are proprietary and usually not available to the public.

1.1 CAN data

Exchange of real-time values between ECUs is done through CAN messages. During a black box security investigation of a modern car, this traffic can be recorded with various tools. A well-known open-source tool to record CAN-communication data is `candump` [2]. A `candump` log entry consists of a timestamp, an interface description, and the CAN-frame with identifier and payload. Listing 1 shows an

[*]Nils Weiß, Enrico Pozzobon, Jürgen Mottok; University of Applied Sciences in Regensburg, Germany, E-mail: nils2.weiss@othr.de, enrico.pozzobon@othr.de, juergen.mottok@othr.de

[†]Václav Matoušek; University of West Bohemia in Pilsen, Faculty of Applied Sciences, Czech Republic E-mail: matousek@kiv.zcu.cz

exemplary output of `candump`. Log files store one CAN message per line. A line contains a timestamp, the interface name, and the CAN message with identifier and data, both in hexadecimal representation. The number sign (#) is used to separate the identifier from the data.

```
(1526379707.348777) can0 22A#8D8802
(1526379707.355973) can0 0C5#F1CD73D3B1D5F242
(1526379707.355975) can0 0D1#C000FFFD00FD00
(1526379707.355975) can0 185#1808
```

Listing 1 *candump log file format output.*

1.2 DBC file format

The DBC file format describes the payload data of a CAN-frame [3]. Listing 2 shows an exemplary data definition for a status message of a battery management system [4]. This example contains the following definitions:

- `BO_` defines a CAN-frame of length 8 with the identifier 341 as data object `BMS_1`.
- This data object contains three different signals, `SG_climit`, `SG_current`, and `SG_soc`.
- The signal `SG_climit` contains the following definitions:
  - `7|8`: This signal starts at bit position 7 and has a length of 8 bits.
  - `@0+`: The endianness of the signal is big-endian and the transferred value is unsigned.
  - `(5,0)`: The transferred value has a scaling factor of 5 and the offset value 0.
  - `[0|35]`: The transferred value has a value range from 0 to 35.
  - `''A''`: The unit of the transferred value is labeled with "A".

On top of the CAN-communication described by DBC files, diagnostic CAN-frames can also be present on the bus. Automotive diagnostic communication is transported via ISO Transport Protocol (ISO-TP) [6]. This ensures are direct communication between two communication partners. Since ISO-TP uses a specific transport protocol and fixed CAN-identifiers for source and destination specification, these messages can be distinguished easily from CAN messages that transport real-time data.

```
BO_341 BMS_1: 8 XXX
  SG_climit :  7|8@0+  (5,0)        [0|35]       "A" XXX
  SG_current: 11|12@0+ (-0.25,500) [-500|1000] "A" XXX
  SG_soc    : 39|16@0+ (0.0025,0)  [0|100]     "%" XXX
```

Listing 2 *DBC file format example.*

1.3 Reverse engineering goals

Open security research on automotive systems often requires knowledge of the transported data on a CAN-bus. Nowadays, no OEM reveals their internal DBC file formats, which would provide the necessary insights into the protocol definitions of the transferred data. Therefore the only way to obtain information about the vehicle internal communication is reverse engineering. In general, reverse engineering is a very time-consuming and therefore expensive task. The following two kinds of information need to be extracted from CAN-traffic, to allow further research on automotive systems:

- The position and format of CAN-signals in a CAN-data field. According to DBC, CAN-signals can start at any bit position and can have a variable length, often not aligned to any usual value, for example, 8, 16, 32. Furthermore, the signals can be encoded either using big-endian or little-endian.
- Every CAN-identifier specifies a DBC signal group, and therefore the kind of transported information. For example, the vehicle speed will always be transferred on CAN messages with a fixed CAN-identifier. A researcher needs to know which CAN-identifiers transports the demanded information. Therefore a mapping of transported information to CAN-identifier is necessary.

Our work provides a statistical reverse engineering method and neural-network-based method to automate the described goals. With statistical methods, we show a possibility to identify data fields in CAN-frames. Neural networks are used to identify the transported real-time data per CAN-identifier. We aim to support research on automotive systems through time savings in the reverse engineering process. Parts of our work are published as open-source software [9].

2. Statistical reverse engineering

Statistical methods are applicable for reverse engineering of a DBC file format from CAN-traffic. This section introduces basic and advanced methods. The goal of statistical reverse engineering is the identification of CAN-signals described in DBC files.

2.1 Feature extraction

The CAN-protocol and the DBC specification allow various early-stage filtering of captured log data. All messages can be grouped by the CAN-identifier since the identifier describes the data transferred in the CAN-frames data payload. Further group-specific features can be extracted. All further explanations are only applied on CAN-frames grouped by their identifier. Useful features are:

- the total number of messages per identifier,
- the number of different values of the CAN-data field,
- the frequency of a CAN message with a specific identifier,

- the variance of the time difference between the transfer of two consecutive CAN-frames with the same identifier.

Only CAN-identifiers that fulfill the following requirements are suitable for an advanced analysis:

- the total number of messages per identifier should be reasonably high, to allow statistical analysis,
- the number of different values of the entire CAN-data field should be reasonably high,
- real-time data is communicated periodically. Higher frequencies indicate more important data,
- the variance of the communication frequency should be very low.

2.2 Transition Aggregation Vector

A powerful feature for statistical analysis of CAN-data is the Transition Aggregation Vector (TAV) of a CAN-identifier group. Let $GF(2) = \mathbb{F}_2 = \{0, 1\} = \mathbb{Z}/2$ be a finite field, containing the elements one and zero. Further, let the vector $\mathbf{x} := (b_{63}, b_{62}, \ldots, b_0) \in \mathbb{F}_2^{64}$ define the data bits of a CAN-frame. Through the selection of a specific CAN-identifier, it's ensured that every CAN-data payload has the same length. A group of CAN-frames with identical CAN-identifier can be represented as a matrix

$$\mathbf{X} = \begin{pmatrix} b_{63,0} & b_{62,0} & \ldots & b_{0,0} \\ b_{63,1} & b_{62,1} & \ldots & b_{0,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{63,n} & b_{62,n} & \ldots & b_{0,n} \end{pmatrix} \in \mathbb{F}_2^{64 \times n}. \tag{1}$$

The matrix $\mathbf{X}$ represents all CAN-data frames of an entire log file. The Transition Aggregation Vector (TAV) of all data bits is computed by the following equation:

$$y_j = \sum_{i=1}^{n} (x_{j,i-1} \oplus x_{j,i}) \qquad y \in \mathbb{N}_0^{64}, \tag{2}$$

where n is the number of messages in a group. j defines the bit index.

2.2.1 Example

Fig. 1 shows the TAV for all CAN messages with identifier `0x3d`. A manual analysis of this TAV shows various observations. The exponential behavior of the transitions in the first 48 bits indicates that integer-like values are transmitted in these bit areas. Another interesting behavior can be observed on bits 53 and 54. Bit 54 has a transition on every CAN message and bit 53 has a transition on every second CAN

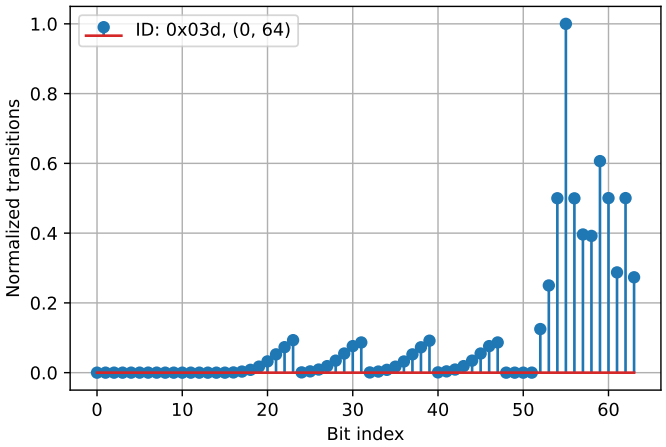


Fig. 1 *TAV of data bits from a group of CAN-frames with identifier* `0x3d`*. The x-axis indicates the bit position inside the CAN-frame. The y-axis indicates the number of bit transitions (bit-flips) between two consecutive CAN-frames.*

message. These two bits, combined in a field, show the behavior of a multiplexer field. This multiplexer manipulates the data represented in the bit areas 55 to 63. For further time series analysis, a demultiplexing into four individual data streams has to be performed. Another indicator that these two bits act as a multiplexer field is the predictability of the bit flips. In vehicular applications, a multiplexer shows the behavior of an incremental counter, which increments on every CAN message. This behavior allows easy identification of multiplexer fields in CAN-traffic. The last remarkable observation of this TAV are constant bits. Constant bits are easy to identify since the number of bit-flips during capture is zero.

2.2.2 Summary

The following observations can be gathered from a TAV:

1. The exponential behavior of neighbored bit transitions indicates integer value fields
$$y_j \approx 2y_{j\pm1}. \tag{3}$$
2. Multiplexer fields have a width between 2 and 4 bits. The least significant bit (LSB) of a multiplexer field has to flip on every CAN-frame. Let n be the number of CAN-frames in an identifier group.
$$y_j \approx n, \quad 2y_j \approx y_{j-1}. \tag{4}$$
3. Constant fields have zero transitions
$$y_j = 0. \tag{5}$$
4. Constant fields and multiplexer fields split data fields.

2.3 Bit-Correlation-Over-Time

All observations on the TAV only consider the bit-flips between two consecutive frames. To fill the gap of the TAV, not be able to take the behavior in a time series into consideration, we introduce a novel method to correlate the behavior of neighboring bits over time. Let

$$\mathbf{A} = \begin{pmatrix} b_{63,0} & b_{62,0} & \dots & b_{0,0} \\ b_{63,1} & b_{62,1} & \dots & b_{0,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{63,n} & b_{62,n} & \dots & b_{0,n} \end{pmatrix} \in \mathbb{F}_2^{64\times n} \tag{6}$$

be a matrix, representing the time series of length n. Each point in time, a row of $\mathbf{A}$, can be represented as a 64-bit vector $\mathbf{v} = (b_{63}, b_{62}, \dots, b_0) \in \mathbb{F}_2^{64}$. As the first operation, the discrete difference over all the 64 columns of $\mathbf{A}$ is computed.

$$b_{j,i} = (a_{j,i} \oplus a_{j,i+1}) \quad {\scriptstyle j=0,\dots,64 \quad i=0,\dots,n-1}. \tag{7}$$

We call the result of this operation matrix $\mathbf{B}$.

$$\mathbf{B} = \begin{pmatrix} b_{63,0} & b_{62,0} & \dots & b_{0,0} \\ b_{63,1} & b_{62,1} & \dots & b_{0,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{63,n-1} & b_{62,n-1} & \dots & b_{0,n-1} \end{pmatrix} \in \mathbb{F}_2^{64\times n-1}. \tag{8}$$

Let m be the convolution length and

$$\mathbf{V} = \mathbb{1} \in \mathbb{F}_2^{64\times m} \tag{9}$$

the convolution matrix of length m. Let's define $\mathbf{b}$ and $\mathbf{v}$ as vectors, representing columns of $\mathbf{B}$ and $\mathbf{V}$. The convolution between $\mathbf{b}$ and $\mathbf{v}$ can be computed through the linear discrete convolution function

$$c_i = (b_i * v_i) \quad {\scriptstyle i=0,\dots,64} \tag{10}$$

The result of equation 10 applied on every column of $\mathbf{B}$ is stored as a matrix

$$\mathbf{C} = \begin{pmatrix} b_{63,0} & b_{62,0} & \dots & b_{0,0} \\ b_{63,1} & b_{62,1} & \dots & b_{0,1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{63,n-1+m} & b_{62,n-1+m} & \dots & b_{0,n-1+m} \end{pmatrix} \in \mathbb{F}_2^{64\times n-1+m}. \tag{11}$$

Let the result of this discrete linear convolution function be the matrix $\mathbf{C}_{64\times n-1+m}$. As the last step, the Pearson correlation coefficient [5] is applied on two consecutive columns of $\mathbf{C}$ to obtain the degree of correlation between two bits.

$$\rho_{c_i,c_{i+1}} = \frac{\operatorname{cov}(c_i, c_{i+1})}{\sigma_{c_i}\sigma_{c_{i+1}}} \qquad i = 0, \dots, 63. \tag{12}$$

The result of this operation is a vector ρ of length 63. Every element of this vector has a value between -1 and 1 and describes the correlation between two consecutive bits over time. Fig. 2 provides an example for messages with CAN-identifier `0x3d`.

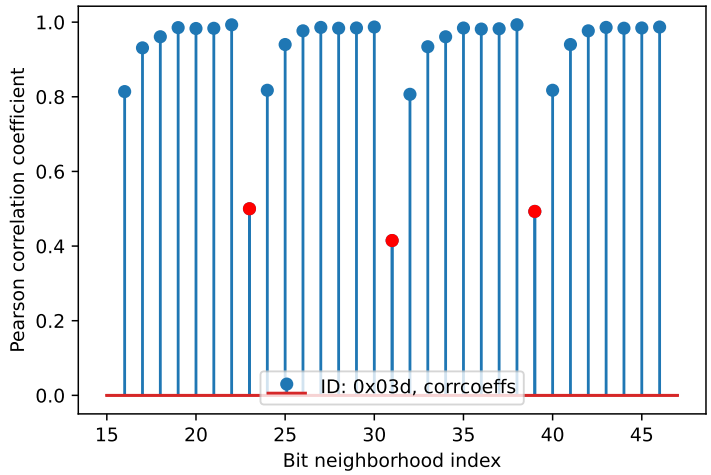


Fig. 2 *Bit-Correlation-Over-Time (BCOT) of all CAN-frames with identifier* `0x3d`. *The x-axis indicates the position between two bits. The represented field is identical to the field shown in Fig. 3.*

2.4 Combination of TAV and BCOT

Since the TAV contains 64 individual data points related to a single bit and the BCOT contains 63 individual data points related to the behavior of two neighboring bits in a CAN-data frame, the derivative of a TAV needs to be computed to combine both metrics. An example is given by the CAN-frames with identifier `0x03d`. The TAV and its derived transition aggregation vector (DTAV) are shown in Fig. 3. An obvious feature of the DTAV are negative values. These negative values are indicators for separated data fields. The example in Fig. 3 shows the TAV of a data field from bit 16 to bit 48. The TAV strongly indicates four different 8-bit wide integer fields. The DTAV (represented from the dashed line) indicates separations after the bits 23, 31, and 39. The combination of both metrics can be used with outlier detection algorithms to identify independent data fields in a CAN-frame. In Fig. 4 a K-means algorithm is used to demonstrate the identification of separators of data fields.

3. Reverse engineering with neural networks

To improve the reverse engineering efforts of CAN-log-data, we evaluate the application of neural networks. This application aims to further increase automated reverse engineering capabilities. While the statistical methods described before were restricted to guessing a field in a single CAN-identifier, another important real-world problem is finding a field in a CAN message belonging to a large number of different CAN-identifiers.

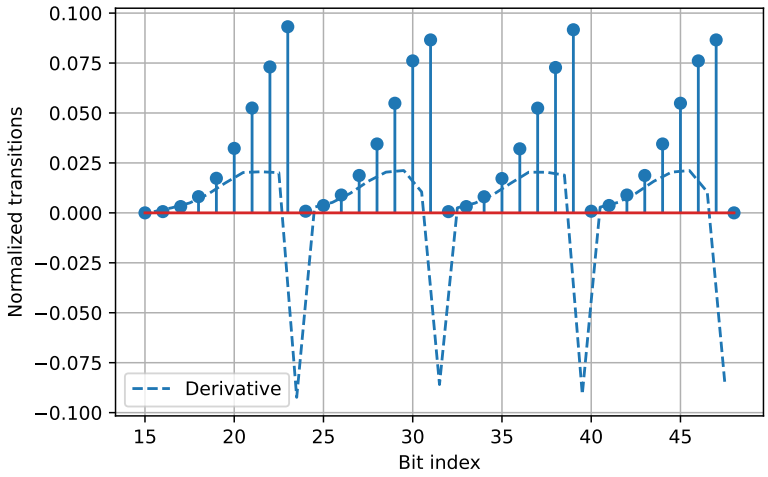


Fig. 3 *The DTAV and normalized TAV of all data bits from the group of CAN-frames with identifier* `0x03d`*. The x-axis indicates the bit position of the CAN-frame. The y-axis indicates the normalized number of transitions. The dashed line marks the DTAV.*

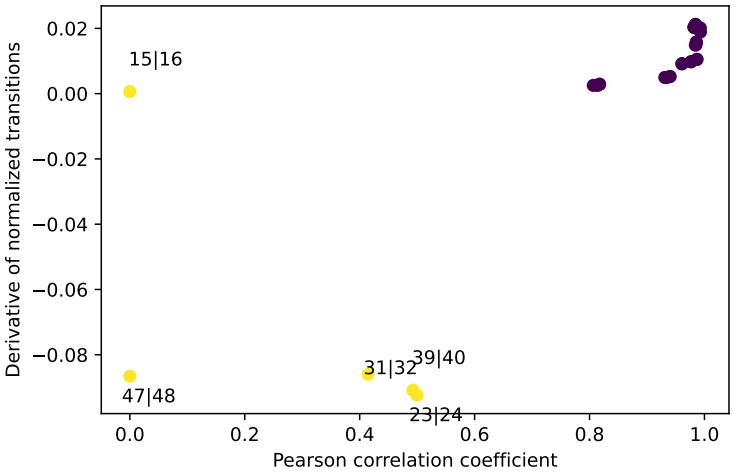


Fig. 4 *Scatter plot of identifier* `0x3d` *with two clusters, computed by a K-means algorithm. The x-axis represents the BCOT. The y-axis represents the values of the DTAV. Yellow data points represent separators for the analyzed field from bit 15 to bit 48. Purple data points are bit neighborhood indices that do not separate a data field.*

3.1 Method Description

A reasonably large capture of CAN-traffic is taken during a real-world driving scenario. Some quantities of the car (e.g. the speed of the car, the pressure on the brake pedal, etc.) are recorded through On Board Diagnostic II (OBD-II) queries [8]. A neural network is then trained to predict an estimation on the known measured quantity given the latest frame of each identifier. To understand which identifiers contain the searched values, two approaches are possible:

1. The weights of the trained network are examined to check which bits of which CAN-identifier have been more important for the correct estimation.

2. Different networks are trained for different CAN-identifiers and the network that obtains the best prediction is associated with the searched identifier.

3.2 Data collection and data-set shaping

The car under test was a Mercedes Class A, year of build 2019. The car was driven for approximately half an hour around the city, while all CAN-frames were captured from the OBD-II connector. All available OBD-II parameter identifiers (PIDs) were also collected, with a sampling rate of approximately 1.3 s^{-1}. In total, the drive took 1700 seconds and $S = 2197$ OBD-II samples were collected for each available PID. In our reverse engineering process, only one measured parameter is considered at a time. For example, let's focus on OBD-II PID `0x0D`, which measures the vehicle speed in km/h. A neural network has the objective to identify which CAN-identifier contains information about the vehicle speed. The value of the measured parameter will be indicated as $y(t)$, where t is the time when the parameter was sampled. The captured CAN-frames were grouped by CAN-identifier, and every constant bit capture is removed from the dataset. All CAN-identifiers that were transmitted with a frequency less than 1 Hz are also ignored. After filtering all rare identifiers and identifiers with constant values, we obtained a dataset of 92 CAN-identifiers with an average of 24 bits used for information encoding. For every arrival time t of sample $y(t)$, a point $x(t)$ is created in the dataset, which includes the last captured CAN-frame for every identifier before time t. Each dataset point $x(t)$ contains $W = 2234$ bits coming from the 92 different identifiers. This preparation step is necessary since the labeled data gathered from OBD-II PIDs have a lower resolution than the captured data from CAN frames. After this preparation, every labeled sample from OBD-II maps to a snapshot containing all current CAN frames. A graphical representation of this dataset and the correlation to the vehicle speed is illustrated in Fig. 5. Different neural networks will be trained to perform an estimation $\hat{y}(t)$ of the value of $y(t)$ given the input $x(t)$, that is, estimate the vehicle speed given at time t given the last CAN-frame received up to that moment on each CAN-identifier. We select three different neural network models for further evaluation. In general, there is a trade-off to make between the complexity of the model and its effectiveness. Given that the number of samples recorded during the drive is quite low, a model with a minimal number of weights should be chosen, to not cause overfitting during the training step.

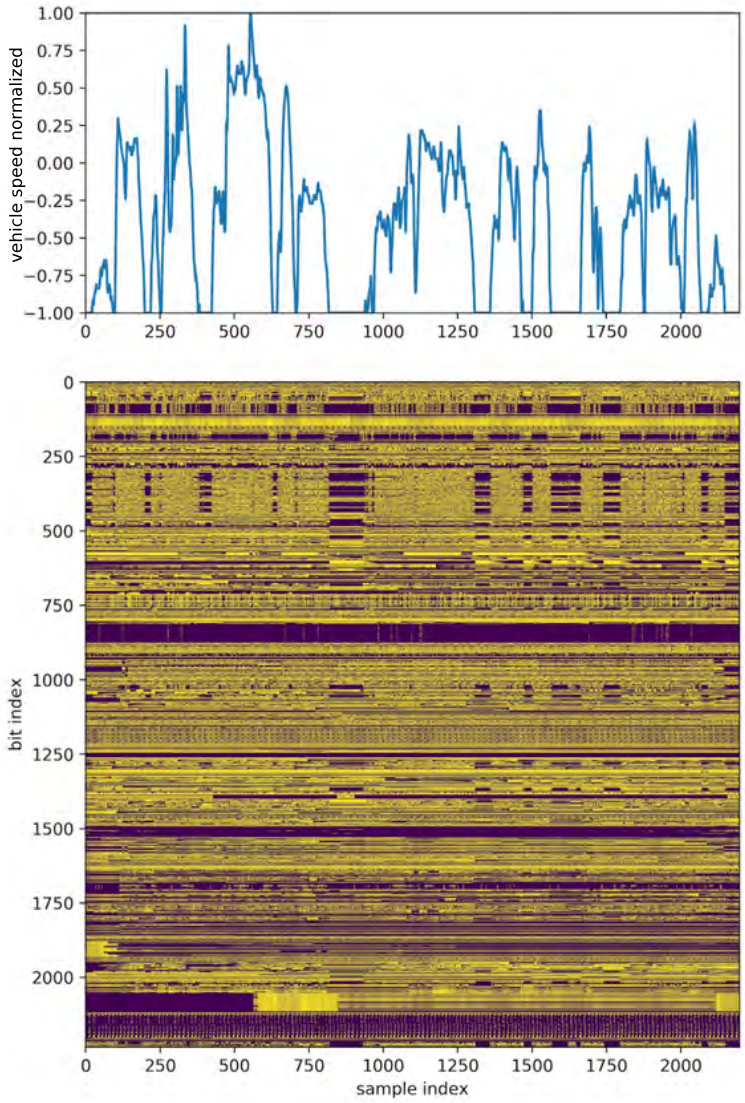


Fig. 5 *Normalized vehicle speed (top) and the states of the bits of the last CAN-frame for each identifier at the arrival times of the samples represented as an image (bottom). In the image, the color yellow represents a bit set to* 1*, and purple represents a bit set to* 0*. Each column of this image is a point* $x(t)$ *in the dataset. It is interesting to note that some bits relate over time to the speed of the car, and are silent when the car is stopped. These bits are good candidates for containing the vehicle speed.*

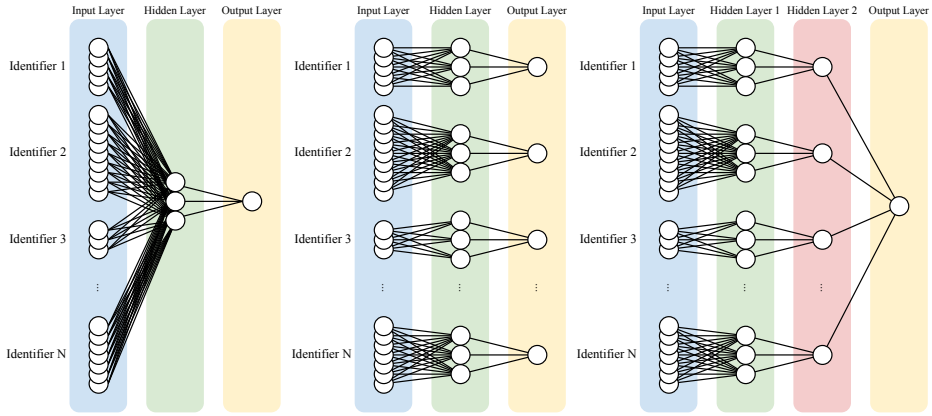
Prevention of Overfitting A big source of overfitting for any neural network attempting this regression is time. Some CAN-identifiers will contain bits that encode the current time or the time since when the car was started. Other identifiers

will contain bits encoding information that directly correlates with time (e.g. engine temperature which will increase over time, GPS coordinates which will change over time, etc.). The side effect of having time-related information encoded in CAN-frames is that the neural network can end up "learning" the speed of the vehicle at every moment in time in the training data and using the time information in $x(t)$ to fit $y(t)$. To avoid overfitting, the neural network will be trained only from samples taken from a continuous-time interval $t_0 < t < t_\mathrm{T}$, and will be tested on a different time interval $t_\mathrm{T} < t < t_\mathrm{END}$. This ensures that any time-related overfitting within the first interval will be detected in the testing phase.

3.3 Description of tested neural networks

Three different neural network topologies were tested for the problem of reverse engineering CAN-identifiers according to the transported information. An overview of these three neural networks is given in Fig. 6. The results of these three neural networks are evaluated by comparing the MAE and RMSE of their predictions [1].

Topology A: Fully connected A first naive approach is to create one big fully connected neural network which includes all bits of all CAN-identifiers as inputs and has a single output (as shown in Fig. 6a). The hidden layer uses the rectified linear unit (ReLU) activation function and the output layer uses a linear activation function. The number of nodes in the hidden layer was empirically chosen to be 12 since it turned out to be a good compromise between training time and the precision of the estimation. Such a network works well enough to estimate the searched value $y(t)$ given the bits $x(t)$, sometimes with a low error. The problem with this network is that guessing which CAN-identifier is containing the searched value is not easy, since this network mixes values from different identifiers.



(a) *Topology A* **(b)** *Topology B* **(c)** *Topology C*

Fig. 6 *Tested neural network topologies.*

Topology B: One network for each CAN identifier A good way to extract the CAN-identifier that carries the desired information $y(t)$ is to train one neural network on each CAN-identifier, and then compare the estimations $\hat{y}_i(t)$ made by each network to $y(t)$ and see which performs better. The network trained on the correct CAN-identifier will be the one with the lowest error. Since the networks are completely separated from each other, it is possible to train them on different systems, leading to a good parallelizability of the algorithm.

Topology C: Linear combination of networks trained on individual CAN identifiers By combining all the networks of the previously described topology B and making a linear combination of their outputs, a new neural network topology is obtained that has a second hidden layer. The activation functions of the second hidden layer and the output layer are linear, meaning that the output is a linear combination of the outputs of the first hidden layer, and therefore the second hidden layer can be eliminated by fully connecting the single output node to the first hidden layer. The identifier that contains the searched information $y(t)$ is found by evaluating all the contributions to the last linear combination and finding which one correlates the most to $y(t)$.

Evaluation Fig. 7 shows the real measured vehicle speed $y(t)$ alongside the estimated vehicle speed $\hat{y}(t)$ from the three described neural network topologies and the estimation error. The training algorithm was executed 100 times for topologies A and C, and 20 times for topology B, since each network in topology B has a much lower number of weights to train. The executions were performed on the first two-thirds of the measurements in the dataset (1464 samples out of 2197 samples). This is visible from the error, which increases after the $\frac{3}{4}$ mark, revealing that some overfitting is happening. When considering the error in the estimations, neural networks with topology A and C perform very similarly and take similar resources for training. Topology A averaged a mean absolute error (MAE) of 0.038 and an root mean square error (RMSE) of 0.057 when a regression is successful. Topology C averaged an MAE of 0.043 and a RMSE of 0.067 when a regression is successful. Topology B outperforms the others in terms of error and immunity to overfitting but required much more computational power since all the individual networks need to be compiled, initialized, and trained separately. However, since it would benefit more in terms of speed from parallelization, it would also be the faster algorithm when distributed to a large number of processors.

For some searched data, neural networks weren't able to find a correlation. This can be caused by two reasons:

1. Only a small number of the measured values with the data present in the CAN-frames could be recorded.

2. Since the CAN traces were recorded from the OBD-II connector, which is connected to only one of many CAN-buses in the car, it is possible that the searched data was present in a different CAN-bus that was not recorded.

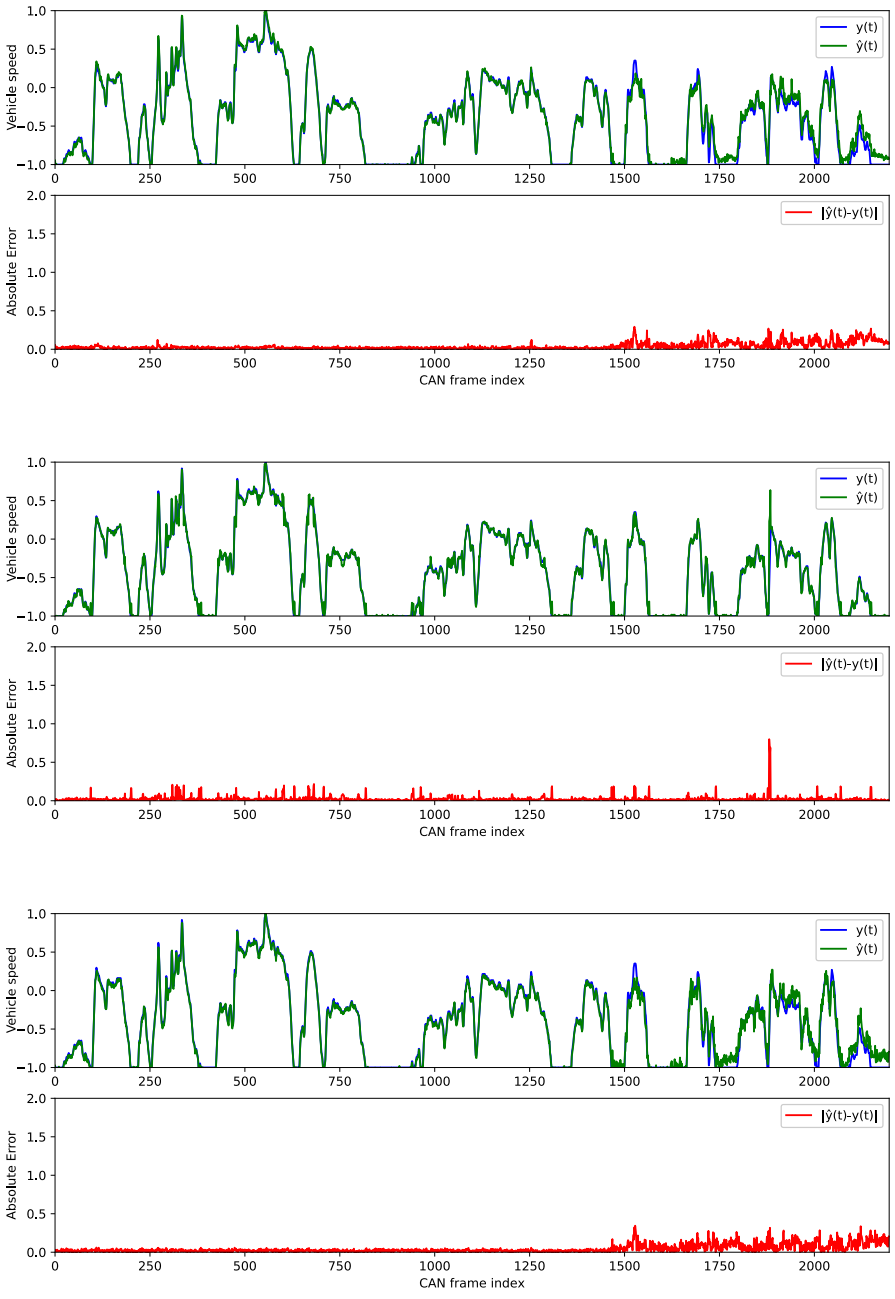


Fig. 7 *Estimation of vehicle speed vs. real vehicle speed and absolute error from the three tested network topologies.*

3.4 Results

As anticipated, the results obtained from neural networks with topology A were not suitable for reverse engineering, since the information from all identifiers was mixed in the hidden layer. By training the multiple subnetworks considered in the topology B, one can guess which identifier contains the searched information by choosing the identifier whose subnetwork gave the least error when testing it against the last third of the dataset. From the results in Fig. 8, it can be seen that CAN-identifier `0x098` is the best match for the vehicle speed, and CAN-identifier `0x087` is a bad match. Interestingly, CAN-identifier `0x14b` estimates the speed somewhat correctly in the training data, but the error increases a lot in the data the network was not trained on, which indicates a case of overfitting.

In topology C, the identifier which contains the searched information is guessed by looking at the intermediate output of each node of the second hidden layer and choosing which one has the highest Pearson correlation coefficient with the searched information. Another equally valid metric is to look at the weights of the last linear combination and choosing the identifier that provides the biggest contribution.

Tab. I shows some of the reverse-engineered CAN-identifiers, along with the RMSE. A lower error means there is higher confidence in the quality of the detection. This demonstrates the application of neural networks with topology B for CAN-protocol reverse engineering.

Searched measurement	**OBD-II PID**	**CAN** id.	**RMSE**
Calculated engine load	`0x04`	`0x0b1`	0.101264
Engine coolant temperature	`0x05`	`0x2b9`	0.051074
Short term fuel trim—Bank 1	`0x06`	`0x0ae`	0.191312
Long term fuel trim—Bank 1	`0x07`	`0x0b1`	0.161904
Intake manifold absolute pressure	`0x0B`	`0x0ae`	0.120086
Engine RPM	`0x0C`	`0x0b1`	0.077556
Vehicle speed	`0x0D`	`0x098`	0.050594
Timing advance	`0x0E`	`0x03d`	0.253927
Intake air temperature	`0x0F`	`0x2b9`	0.175567
Throttle position	`0x11`	`0x03d`	0.096419
Fuel Rail Gauge Pressure	`0x23`	`0x147`	0.135893
Relative throttle position	`0x45`	`0x03d`	0.108917
Ambient air temperature	`0x46`	`0x339`	0.034543
Absolute throttle position B	`0x47`	`0x03d`	0.116226
Accelerator pedal position D	`0x49`	`0x03d`	0.071383
Commanded throttle actuator	`0x4C`	`0x03d`	0.096537

Tab. I *Results of reverse-engineering of CAN-identifiers through neural networks with topology B.*

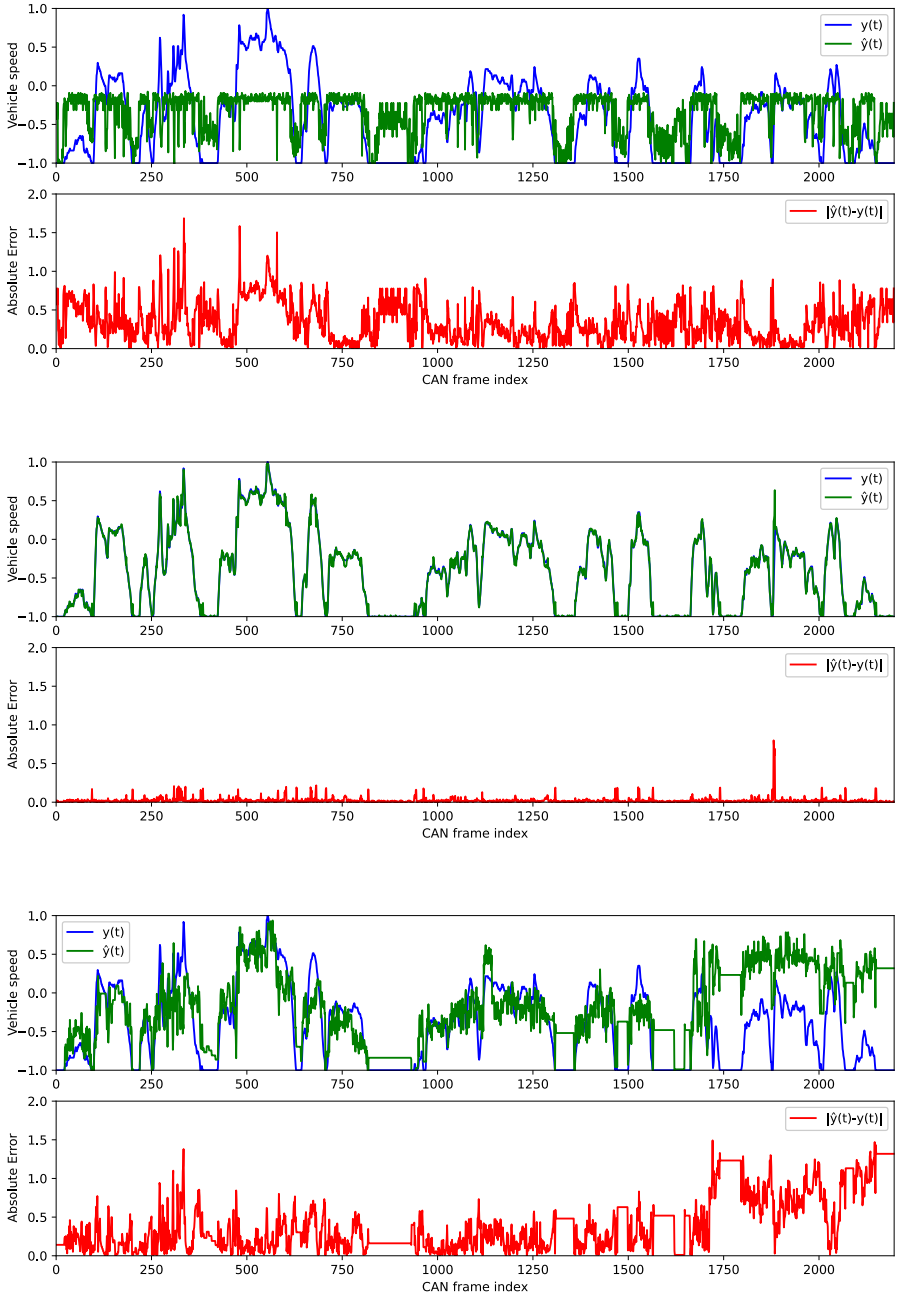


Fig. 8 *Estimation of vehicle speed vs. real vehicle speed and absolute error from the three subnetworks in topology B, each associated with three different identifiers.*

4. Conclusion

This article shows the capabilities of statistical and neural network reverse engineering methods. We demonstrated the localization of data fields within a CAN-frame through advanced statistical methods. All examples were performed on real-world data, captured during a 30-minute test drive.

TAV, DTAV, and BCOT are statistical metrics that can identify individual data fields in a CAN-frame. This identification is an important step in the reverse engineering process. For further research involving time-series analysis, all individual fields in a CAN-frame need to be known.

Neural networks are a valid tool for reverse engineering a proprietary CAN-bus protocol. While a conventional approach like an exhaustive search of the fields and correlation of the searched measurements with their values would also work and could be optimized to use less computational power, the advantage of neural networks is that they reduce development time since the algorithm is already implemented in the library and the only work necessary is building the dataset and specifying the network topology. The performance of the presented algorithm could be improved by using more specific neural network models, for example recurring neural networks, which are specialized for processes that evolve in time. However, the naive presented approach was able to produce usable results with an extremely small dataset, and using more advanced types of neural networks would surely require recording measurements for a much longer time.

Acknowledgement

The present work as part of the PetS^3 project was funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (Bayerisches Staatsministerium für Wirtschaft, Landesentwicklung und Energie) under grant number IUK-1711-0018. The authors are responsible for the content of this publication. Furthermore, this research was conducted during the doctoral studies of Nils Weiß and Enrico Pozzobon at the Faculty of Applied Sciences, University of West Bohemia in Pilsen, CZ.

References

[1] BISHOP C.M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, doi: 10.5555/1162264.

[2] HARTKOPP O. SocketCAN userspace utilities and tools [software]. 2021-06-28 [accessed 2021-07-09]. Available from: https://github.com/linux-can/can-utils

[3] VECTOR INFORMATIK GMBH *DBC File Format Documentation*. Vector Informatik GmbH, 2008.

[4] Open Vehicles Monitoring System: *DBC Introduction* [online]. Open Vehicles Developers Revision 5565dbf9 [viewed 2021-07-09]. Available from: https://docs.openvehicles.com/en/latest/components/vehicle_dbc/docs/dbc-primer.html

[5] PEARSON K. Notes on Regression and Inheritance in the Case of Two Parents. In: *Proceedings of the Royal Society of London*, 1895, pp. 240–242, doi: 10.1098/rspl.1895.0041.

[6] WEISS N., RENNER S., MOTTOK J., MATOUŠEK V. Automated Threat Evaluation of Automotive Diagnostic Protocols. In: *Proceedings of the Embedded Security in Cars Workshop (ESCAR)*, Virtual, 2021.

[7] ISO Central Secretary. ISO 11898-1:2015: *Road vehicles – Controller area network (CAN) — Part 1: Data link layer and physical signalling* [online]. International Organization for Standardization. 2015 [viewed 6 July 2021]. Available from: https://www.iso.org/standard/63648.html

[8] ISO Central Secretary. ISO 27145-3:2012: *Road vehicles – Implementation of World-Wide Harmonized On-Board Diagnostics (WWH-OBD) communication requirements – Part 3: Common message dictionary* [online]. International Organization for Standardization. 2012 [viewed 6 July 2021]. Available from: https://www.iso.org/standard/46277.html

[9] EMUNDO GMBH. revdbc [software]. 2020 [accessed 2021-07-15]. Available from: https://github.com/emundo/revdbc