Affordable and Effective Classification of ECU Signal Source using k-Nearest Neighbors and Naive Bayes for Identity Confirmation

Joel Valleroy
ECE Department
University of Michigan
Dearborn, USA
jovaller@umich.edu

Daniel Beier
CECS Department
University of Michigan
Dearborn, USA
dbeier@umich.edu

***Abstract*—The modern automobile has dozens of ECU (electric control units), with multiple often broadcasting across the same network. Also in modern times, it has become apparent that criminals with nefarious tools could potentially take control of a modern automobile remotely. Counterfeit and illegitimately installed ECUs could also be a concern, causing hostile or otherwise illegitimate behavior in controllers important for safety. Due to these concerns, the need for additional security in modern automobiles has become readily apparent. The need for additional cyber security is especially true given that future vehicles are becoming increasingly automated.**

In this paper, we demonstrate two applied methods of machine learning for classifying ECU signals, utilizing a carefully engineered feature set extracted from samples from eight different ECUs. Our two chosen machine learning methods for demonstrating our application here are k-nearest neighbors and naive bayesian.

***Index Terms*—machine learning (ML), k-Nearest Neighbors (kNN), naive Bayes, cyber security, automotive security, CAN (controller area network), ECU (electronic control unit).**

I. Introduction

Automobiles in recent times utilize numerous controllers (ECUs) to control various features of the vehicle. These ECUs communicate with eachother across mediums such as CAN (controller area network). The importance of each ECU ranges from low to critical, for example, one may control the brakes, while another may only monitor tire pressure.

The cyber security preparedness of modern vehicles has become under increased scrutiny under the last few years, after it became apparent there was none to little security beyond the air gap (no remote connectivity), and future cars are becoming more connected, or not-air-gapped. While innovations such as self-driving cars that communicate with each other are amazing and worthwhile, they further increase the need for security.

Because of the need for further security, we present in this paper two applied methods for classifying ECU signals, and a useful set of features that we use to accomplish this classification. We look forward to further derivative works and advancements in the field of automotive internal security.

II. Related Works

Recently (2019), Hafeez et. al. [1] demonstrated an applied security technique where they captured time-series voltage output from several ECUs communicating over CAN, extracted several relevant features, and trained a neural network to classify the source of a sample signal, that is, identify which ECU it came from. They found high quality results from this method, 97.4%.

Utilizing a similar but simpler approach, our work in this paper shows that almost as effective, 90%, results can be obtained utilizing human comprehensible machine learning methods such as k-Nearest Neighbors and Naive Bayes in conjunction with effective feature extraction.

In 2017, Avatefipour et al. [2] also demonstrated a similar approach to tackle ECU fingerprinting. This was conducted in a similar manner to our approach in which different features were found and utilized from the CAN High voltage. The difference between their technique and ours is that they also incorporated differentiating CAN channels. This shows that this type of application could be viable in a plethora of different domains.

In 2018, Hafeez et. al. [3] showed that when utilizing frequency response of different ECU's transmitting the same data package, it was possible to determine the source ECU. This was due to every ECU having a slightly different frequency response. This data was then utilized to train and test a neural network which produced 94% accuracy rating.

In 2019, Tian et. al. [4] showed that the temperature variations prove to be an issue with fingerprinting different ECU signals. This is due to the variance in the clock offset. They found that the average clock offset of different ECUs were about the same when tested at the same temperature, whereas when the temperature varies the clock offset also varied. This was shown utilizing some ECUs at 20°C while

some were ran at 0°C. It raises some concerns about other fingerprinting techniques as if this is not taken into account during development, fingerprinting techniques could fail.

In 2018, Choi et. al. [5] conducted a similar approach as our project although when doing feature extraction, they utilized only the extended identifier field of a CAN message. Doing this they utilized SVM, NN, and BDT. While doing this it was noticed that better results were obtained when there are more dominant states than recessive (i.e. more 0's than 1's).

III. System Modeling

For purposes of proof of concept, we begin with a collection of gathered signal samples from 8 separate ECUs.

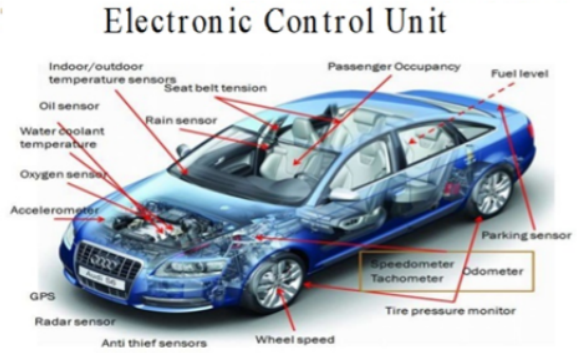

Fig. 1. ECU layout example - Source: Adapted from [6]

The signal samples consist of a rectangular wave, repeated for several cycles. Due to various electrical engineering manufacturing accuracy, the signals which have variances between them - that is - depending on where they are sourced from. Utilizing a technique called fingerprinting, we will attempt to extract relevant features from a given signal sample and then utilize machine learning methods to classify, or identify, the source of said signal sample.

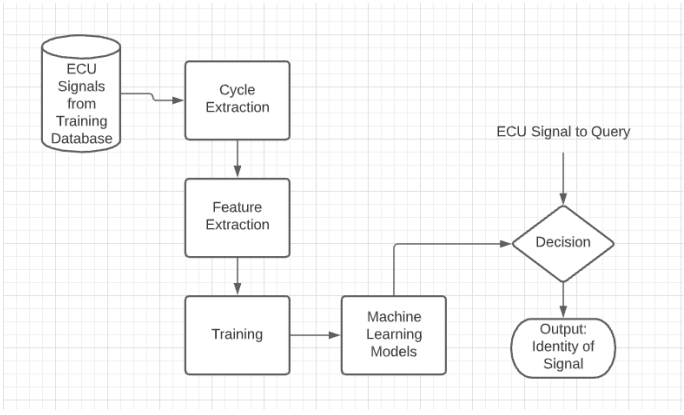

Fig. 2. System Architecture

A. Cycle Extraction:

Before we can start extracting features, we must perform at least some pre-processing. We elected to break the samples down to the cycle level because each sample consisted of a cycle repeated several times. This implementation effectively quintupled our available data samples to train with.

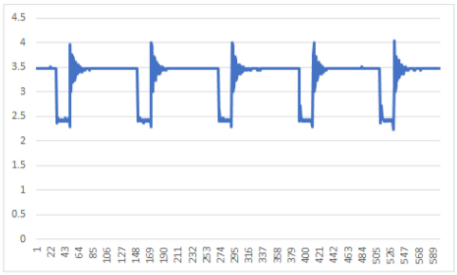

Fig. 3. Data Sample

Fig. 4. Extracted Cycle

B. Feature Selection:

After pre-processing, we isolate 6 different features from each sample. These features are: duty cycle, voltage minima, voltage range, linear discriminant analysis (LDA), independent component analysis (ICA), and principle component analysis (PCA).

Duty Cycle

$$\frac{\sum_{k=0}^{n}(1 \text{ if } sample[k] > 3V, else\ 0)}{n} \quad (1)$$

Duty Cycle shown in equation (1) works by examining the signal in question and if it is found to be greater than 3V then it is considered a 1, if it is lower it is considered a 0. This is carried out until all of the samples within the particular class is completed, adding together the outcomes of the comparisons, and finally dividing this by the total number of samples within the class. The following results were produced after finding the Duty Cycle of each individual class, as shown in the density graph figure (5).

Voltage Minima

$$\arg\min\ (V) \quad (2)$$

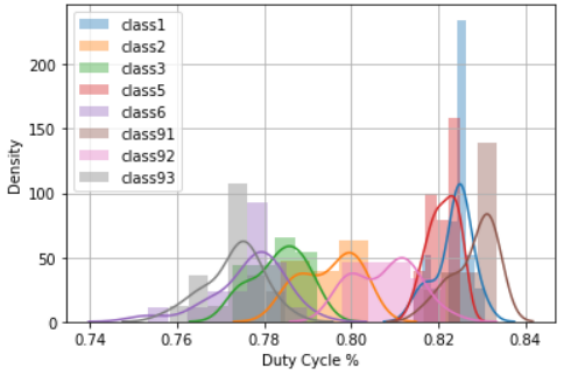

Fig. 5. Duty Cycle %

Our voltage minima feature simply represents the lowest point in a queried cycle. The location of this point as relevant to the rectangle wave can vary depend on the ECU it was sourced from.

The following results were produced after finding the voltage minima of each individual class, as shown in the density graph figure (6).

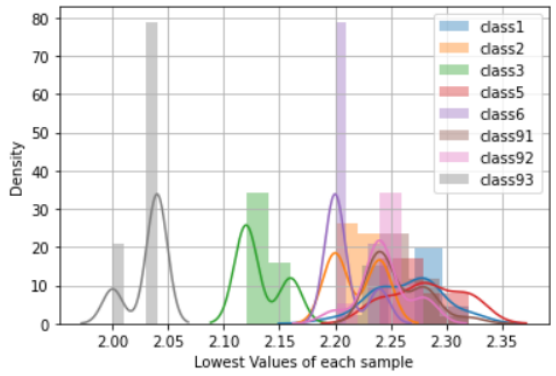

Fig. 6. Voltage Minima %

Voltage Range

$$\arg\max\ (\mathrm{V}) - \arg\min\ (\mathrm{V}) \tag{3}$$

Our voltage range feature represents, in a given cycle, the greatest peak subtracted by the lowest minima.

The following results were produced after finding the voltage range of each individual class, as shown in the density graph figure (7).

Linear Discriminant Analysis (LDA)

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + log(\pi_k) \tag{4}$$

Linear Discriminant Analysis (LDA) shown above in equation (4) works by assuming that there is an equal covariance

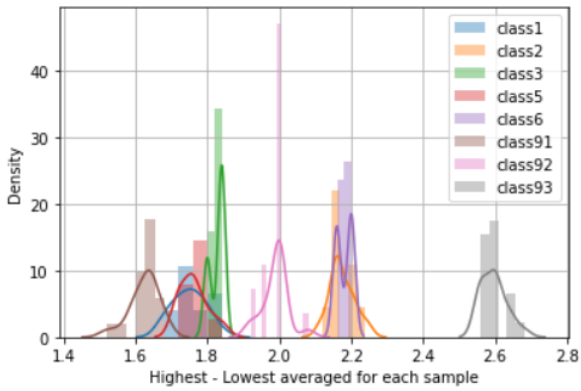

Fig. 7. Voltage Range %

(Σ) amongst the classes. You may then utilize this information along with the mean (μ) of the data (x) to find the highest likelihood amongst the classes. The following results were produced after finding the LDA of each individual class, as shown in the density graph figure (8).

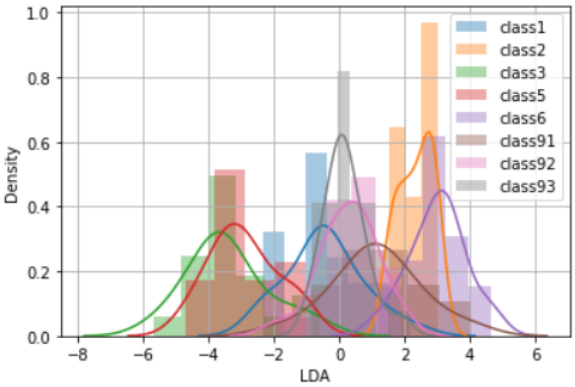

Fig. 8. LDA Distribution

Principle Component Analysis (PCA) is utilized to reduce the dimensionality of variable space. This is carried out by the following 5 steps:

1) Scale the data. This is carried out by subtracting the mean from the data and then dividing by the standard deviation.
2) Calculate the covariance matrix.
3) Calculate the eigenvectors and eigenvalues.
4) Find the principle components. This is done by sorting the eigenvectors by diminishing eigenvalues and pick k eigenvectors with the biggest eigenvalues.
5) Determine the new pivot by re-direction of the data according to the principle components.

The following results were produced after finding the PCA of each individual class, as shown in the density graph figure (9).

Independent Component Analysis (ICA) is utilized to isolate a multivariate signal into its basic segments. This is carried out

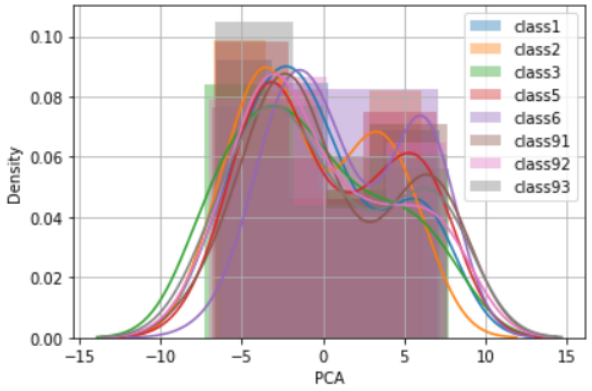

Fig. 9. PCA Distribution

by the following 7 steps.

1) Center the data. This is carried out by subtracting the mean from the data.
2) "Whiten" the data by changing it so that expected relationships between its segments are eliminated (covariance equivalent to 0) and the fluctuation of every part is equivalent to 1.
3) Pick an arbitrary introductory value for the de-blending matrix w.
4) Find the updated value for w.
5) Compute the normalization of w.
6) Check whether calculation has converged and in the event that it hasn't, get back to stage 4.
7) Utilize the dot product of the originating data and matrix w. This leaves you with the independent signals.

The following results were produced after finding the ICA of each individual class, as shown in the density graph figure (10).

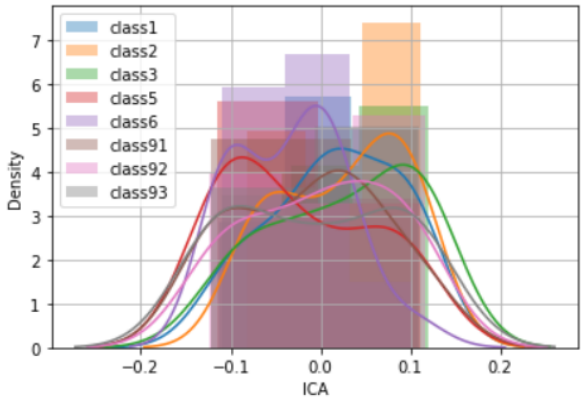

Fig. 10. ICA Distribution

IV. Methodology

A. k-fold Cross-Validation:

To prevent overfitting on specific training segments, a technique such as k-fold Cross-Validation is necessary. This technique takes your dataset and insures the training segments and the validation/testing segments are different each run, and that the results are recorded after and averaged.

We implemented a more robust version of k-fold Cross-Validation by using a shuffling technique. Every time we train and test a new model, the dataset is randomly shuffled before the training and testing segments are divided. This allows us to ensure that there is no overfitting present.

B. k-Nearest Neighbors:

k-Nearest Neighbors is a machine learning algorithm that, in its simplest format, finds the k (where k is a number) nearest neighbors to the query point, and tallies up their vote on what class the query point is likely to be.

This is done by first computing the distance of every training point, from the query point. The distance formula for two dimensions is as follows:

$$distance = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2} \tag{5}$$

Once you have all the distances, you then sort by distance - smallest values first.

Depending on the value of k chosen, let us assume 3, you will examine the 3 closest neighbors to the query point. Taking each close neighbor's class, you will tally up the results. The class with the most votes will be the assumed class of the query point.

For a k of 1, you just need to examine the closest neighbors class to classify the query point, that is:

$$class_(query) = class_(closest\ neighbor) \tag{6}$$

This algorithm can become more advanced with a weighting function. A typical weighting function is distance based, that is the close neighbors that are closer to the query point will be worth more than the close neighbor that is the farthest.

For our kNN method, we utilized our first 3 features, that is: duty cycle, voltage minima, and voltage range.

C. k-Nearest Neighbors Hyper Parameter Tuning:

k-Nearest Neighbors systems can be hyper tuned by adjusting the amount of neighbors considered, and the weighting algorithm (if any).

Since adjusting these is cheap, we decided to check several different tuning each run. We test with 6 different models, with values of k between 3 and 7, and an additional weighting function applied as well.

D. Naive Bayes:

The Naive Bayes algorithm is based upon the Bayes Theorem. This theorem utilizes prior conditions that could be related to an event in order to determine the probability of an event occurring.

Bayes Theorem

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{7}$$

The above equation (7) shows how Bayes Theorem is carried out. The posterior probability is shown as P(c|x) and represents the probability of a class given a particular predictor. This is found by first multiplying the likelihood of P(x|c) with the classes prior probability P(c). From here you must divide the outcome of this by the predictors prior probability P(x).

When utilizing this algorithm within our project the python library takes a similar approach to tackle this equation. This is carried out utilizing the Gaussian Naive Bayes equation as shown below in equation (8).

Gaussian Naive Bayes

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\Big(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\Big) \tag{8}$$

Equation (8) above is carried out in a similar manner as equation (7) as the σ_y and μ_y are estimated by utilizing the likelihood. For our implementation the model was trained with our selected features and classifications of the features. This was conducted utilizing a combination of different feature sets in order to compare the accuracy ratings of different combinations.

V. Results

For our results section, we will compare the accuracy metrics between our two methods - including their tuned versions. Figure (12) and (13) show our results across every model. These results shown are the percent accuracy that each model found.

For k-Nearest Neighbors, we found in our five runs that the simplest version (k = 3, no weights) had the best results and lowest deviation. It was more much more consistent with its average of 88% than other versions, despite other versions occasionally seeing accuracy as high as 95%. These k-nearest neighbors runs were achieved using our features duty cycle, absolute minima, and voltage range.

For our Naive Bayes models, we utilized four different feature sets. NB1 used Duty Cycle, Absolute Minima, and Voltage Range. NB2 used LDA, Duty Cycle, Absolute Minima, and Voltage Range. NB3 used ICA, Duty Cycle, Absolute Minima, and Voltage Range. NB4 used PCA, Duty Cycle, Absolute Minima, and Voltage Range.

In the case of Naive Bayes, NB1 and NB3 performed the best, with an average accuracy over 5 runs of 86.6%.

So, at least in the case of these 5 test runs and with our chosen feature sets, the most simple model (kNN with k of 3) achieved the best results.

Another interesting note is that when executing NB with either PCA as a feature set or ICA as a feature set, the accuracy rating was the same (shown in figure 11). This is interesting due to the fact of how PCA and ICA are polar opposites in the way in which the algorithm works. Where PCA focuses on maximizing the variance whereas ICA doesn't focus variance. It is believed to be this way due to the repetitive data points that are incorporated within this data set but further investigation is warranted.

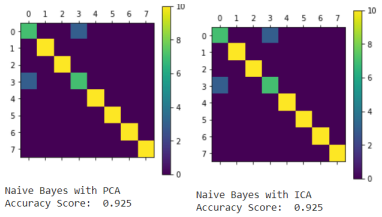

Fig. 11. PCA vs ICA

Lastly it was noticed that the Naive Bayes method took on average less time to carry out than K-Nearest Neighbors. This was observed by implementing print statements around the training and predicting. This is most likely due to the larger amount of computation required to carry out K-Nearest Neighbors. Whereas the timings were not very far apart you can see in figure (14) that Naive Bayes method was on average faster and thus should be utilized over K-Nearest Neighbors.

VI. Conclusion

In this paper, we presented a simple fingerprinting method for ECU signals that still achieved close to 90% accuracy. We did this by utilizing step extraction, feature engineering, and two machine learning methods - k-Nearest Neighbors and Naive Bayes. The motivation was cyber security. Feature engineering is perhaps the most important part as all other results are derived from the quality of that. Features in general must be appropriate to distinguish classes. Our chosen features were based on electrical engineering manufacturing differences in ECU which resulted in different outputs. k-Nearest Neighbors and Naive Bayes, our chosen methods, were chosen specifically for their known usefulness and comprehensibility. As such, we were able to show that one does not need a convoluted convolution neural network to achieve reasonable results even with live data. We greatly look forward to derivative works and any further improvements or deployments in the field of automotive security.

VII. References

[1] A. Hafeez, K. Topolovec and S. Awad, "ECU Fingerprinting through Parametric Signal Modeling and Artificial Neural Networks for In-vehicle Security against Spoofing Attacks," 2019 15th International Computer Engineering Conference (ICENCO), 2019, pp. 29-38, doi: 10.1109/ICENCO48310.2019.9027298.

[2] O. Avatefipour, A. Hafeez, M. Tayyab and H. Malik, "Linking received packet to the transmitter through physical-fingerprinting of controller area network," 2017 IEEE Workshop on Information Forensics and Security (WIFS), 2017, pp. 1-6, doi: 10.1109/WIFS.2017.8267643.

[3] A. Hafeez, M. Tayyab, C. Zolo and S. Awad, "Finger Printing of Engine Control Units by Using Frequency Response for Secure In-Vehicle Communication," 2018 14th International

	k-Nearest Neighbors						Naïve Bayes			
k-fold	k=3	k=3, w	k=5	k=5,w	k=7	k=7,w	NB_1	NB_2	NB_3	NB_4
1	88	83	94	88	95	86	91	78	90	90
2	86	85	84	84	83	84	88	74	86	86
3	88	86	85	86	79	85	83	81	83	83
4	90	89	85	86	83	88	88	81	89	83
5	88	85	80	84	81	83	83	68	85	85
avg	88	85.6	85.6	85.6	84.2	85.2	86.6	76.4	86.6	85.4
sd	1.264911	1.959592	4.586938	1.496663	5.6	1.720465	3.136877	4.923413	2.57682	2.57682

Fig. 12. Results Table As Percent

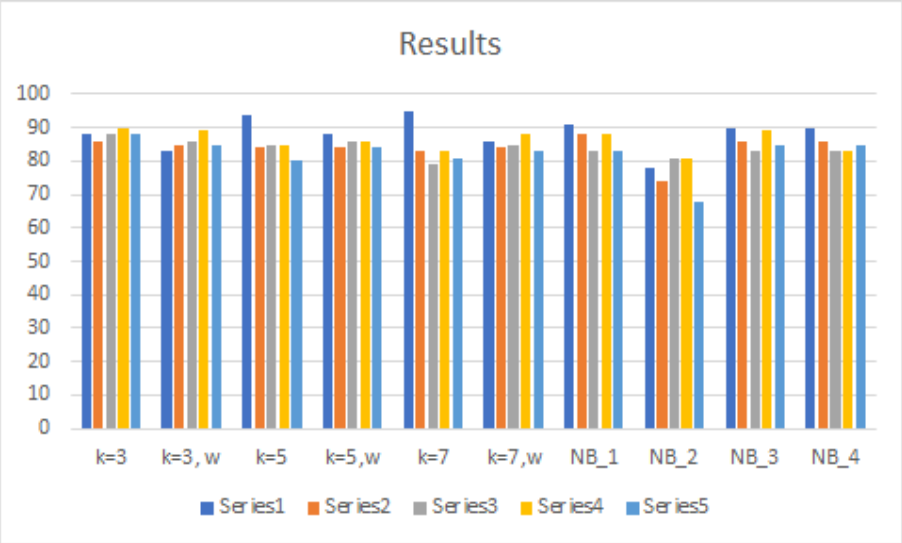

Fig. 13. Results Chart As Percent

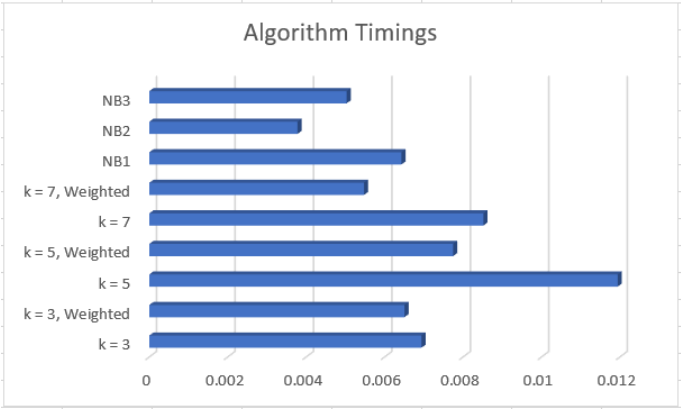

Fig. 14. Algorithm Timings

Computer Engineering Conference (ICENCO), 2018, pp. 79-83, doi: 10.1109/ICENCO.2018.8636140.

[4] M. Tian, R. Jiang, C. Xing, H. Qu, Q. Lu and X. Zhou, "Exploiting Temperature-Varied ECU Fingerprints for Source Identification in In-vehicle Network Intrusion Detection," 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019, pp. 1-8, doi: 10.1109/IPCCC47392.2019.8958766.

[5] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park and D. H. Lee, "Identifying ECUs Using Inimitable Characteristics of Signals in Controller Area Networks," in IEEE Transactions on Vehicular Technology, vol. 67, no. 6, pp. 4757-4770, June 2018, doi: 10.1109/TVT.2018.2810232.

[6] Photograph of ECU Layout, ECU'S, Accessed on: August 12, 2021. [Powerpoint] ECE 5831 Project.ppt