Automotive Control using Surface Electromyography

Gia Quoc Bao Tran

11 September 2020
(updated January 2021)

Submitted to the
Institute of Engineering, Université Grenoble Alpes
and the
Institute of Industrial Science, The University of Tokyo
as a final report of
Research Internship
at the
K. Nakano Laboratory, Institute of Industrial Science, The University of Tokyo
June – September 2020

Written by
Gia Quoc Bao Tran
Master of Engineering Student, Institute of Engineering, Université Grenoble Alpes
Research Intern

Certified by
Kimihiko Nakano
Professor, Institute of Industrial Science, The University of Tokyo
Research Supervisor

Abstract

Advanced driver-assistance systems have proven their capability of assisting drivers in certain aspects, such as driving safety or comfort. In an attempt to help the (partially) disabled to drive, relieve long-distance driving fatigue, as well as reduce risks of muscle injuries, we seek to replace the conventional automotive control interface with a modern human-machine interface through the use of surface electromyography (sEMG) sensors attached to the driver's muscles. In this study, we propose several aspects of using sEMG-controlled interfaces for driving assistance, particularly for braking control of vehicles.

We build a database of sEMG signals measured from the activities of five leg muscles corresponding to the driver's braking and preparing for braking. Power spectral analysis is first performed to identify the components of these signals, which include noise from the electrical network (periodic, 50 Hz), from the driver's heartbeat, and unknown noise. Different signal processing methods are proposed and tested, namely the traditional and optimal linear (Wiener) filtering algorithms and independent component analysis, which allow us to separate the independent components in a set of signals.

We then develop and test, by simulation, different machine learning-based classification algorithms, including the k-nearest neighbors, deep reinforcement learning, and deep learning using a bidirectional long short-term memory network with extracted features as input to classify the preprocessed signals into two groups: braking and not braking the vehicle. This output is used for vehicle braking control.

Keywords: advanced driver-assistance systems (ADAS), human-machine interface (HMI), surface electromyography (sEMG), independent component analysis (ICA), machine learning, long short-term memory

Résumé

Les systèmes avancés d'assistance à la conduite ont prouvé leur capacité à aider les conducteurs dans certains aspects, tels que la sécurité de conduite ou le confort. Dans le but d'aider les personnes (partiellement) handicapées à conduire, de soulager la fatigue de conduite sur de longues distances et de réduire les risques de blessures musculaires, nous cherchons à remplacer l'interface de contrôle automobile conventionnelle par une interface homme-machine moderne grâce à l'utilisation des capteurs d'électromyographie de surface (sEMG) fixés aux muscles du conducteur. Dans cette étude, nous proposons plusieurs aspects de l'utilisation des interfaces contrôlées par sEMG pour l'assistance à la conduite, en particulier pour la commande de freinage des véhicules.

Nous construisons une base de données de signaux sEMG mesurés à partir des activités de cinq muscles des jambes correspondant au freinage du conducteur et à la préparation du freinage. Une analyse spectrale de puissance est d'abord effectuée pour identifier les composants de ces signaux qui incluent le bruit du réseau électrique (périodiques, 50 Hz), du rythme cardiaque du conducteur et le bruit inconnu. Différentes méthodes de traitement du signal sont proposées et testées, à savoir les algorithmes de filtrage traditionnel et linéaire optimal (Wiener), ainsi que l'analyse en composantes indépendantes qui permet de séparer les composantes indépendantes contenues dans un ensemble de signaux.

Nous développons et testons ensuite, par simulation, différents algorithmes de classification basés sur l'apprentissage automatique, y compris les k voisins les plus proches, l'apprentissage par renforcement profond et l'apprentissage profond à l'aide d'un réseau avec mémoire à long-court terme bidirectionnel avec des caractéristiques extraites comme entrée pour classer les signaux prétraités en deux groupes: freiner et ne pas freiner le véhicule. Cette sortie sera utilisée pour la commande de freinage du véhicule.

Mots clés: systèmes avancés d'assistance à la conduite, interface homme-machine (IHM), électromyographie de surface (sEMG), analyse en composantes indépendantes, apprentissage automatique, mémoire à long-court terme

Acknowledgements

I would like to thank my supervisor, Professor Kimihiko Nakano, for his advice during the internship. He was friendly and highly supportive, even when the situation prevented me from coming to Japan.
Thanks to all the researchers at the K. Nakano Laboratory who worked with me on the project. Koge-san helped me do the experiments, Wang-san helped me familiarize myself with reinforcement learning, and Yang-san gave me his useful advice.
Thanks to my professors at the Institute of Engineering, Université Grenoble Alpes, especially Professor Olivier Sename, who taught me Automatic Control and its applications to autonomous vehicles.
Thanks to the Honda Foundation, which agreed to fund my internship in Tokyo fully. Even though the difficult global situation prevented that from happening, they still supported me the most they could and agreed to save the funding so that I can come to Japan in the future. Thanks also to their employees in Vietnam and Japan, and to the administrative staff at The University of Tokyo for all the support. I will certainly come to Japan after graduating.
Thanks to my family and friends for all the support and for motivating me in such a difficult time. I know you are in a difficult time too, and I hope you can overcome this as I did.
Finally, thanks to myself for not giving up the internship despite the situation. Working at a distance is itself a skill that each of us would need, not just in this special time, but in all our lives.
Thank you. ありがと. 谢谢. Merci.

Acronyms

The meaning of an acronym is indicated once, when it first appears in the text.

DDPG	deep deterministic policy gradient
DL	deep learning
DNN	deep neural network
DRL	deep reinforcement learning
FIR	finite impulse response
HMI	human-machine interface
ICA	independent component analysis
IIR	infinite impulse response
k-NN	k-nearest neighbor
LSTM	long short-term memory
ML	machine learning
PSD	power spectral density
RL	reinforcement learning
RMS	root mean square
RNN	recurrent neural network
ROI	region of interest
sEMG	surface electromyography

List of Figures

List of Tables

Contents

1 Introduction

1.1 Overview and state of the art

This study is about controlling an automated vehicle using surface electromyography (sEMG) signals obtained from electrodes attached to the driver's muscles. These signals contain information about the movements of the driver's muscles, which are used to control the vehicle accordingly. The motivations for this are [1-4]:

- To enable the disabled to drive a car: the electrodes allow for automotive control through a human-machine interface, so a person with one active arm and leg can still operate the vehicle.
- To release long-distance driving fatigue: this strategy requires less energy, as we do not rely on the force exerted by the body.
- To reduce the risks of injuries when doing U-turns [5]: there is a risk of shoulder muscle injuries when we turn the steering wheel at large angles. This strategy can help prevent that from happening.

The use of electrodes requires less force than other devices, e.g., joysticks [6] or strain gauges [7], and thus can be more suitable for people without health conditions. Previous studies include the introduction of an sEMG-based driving assistance interface for automotive steering control and the test with a driving simulator [1-4]. In these studies, the authors designed and evaluated different strategies of steering assistance using sEMG signals as an interface. In this work, we focus on automotive braking control using sEMG signals. By attaching several electrodes to the driver's leg muscles, we record the activities of these and control the vehicle accordingly. The method is to use machine learning (ML) to help the vehicle decide whether these leg muscle activities correspond to braking the vehicle or not (binary classification problem). As ML-based methods typically require well-processed data, the method used to denoise the obtained signals is crucial. The work is, therefore, divided into two main parts: signal processing and signal classification using ML.
For signal processing, the author proposes and compares three approaches. First, we discuss a simple filtering method using a traditional bandstop filter, which is later shown to cause a considerably large delay in the filtered signal. A second approach using optimal linear filtering is then proposed, which results in a much smaller delay. The third method is not filtering, but instead, it separates the components contained in the signal using an algorithm called independent component analysis (ICA). By this, we separate the noises from the main, useful part of the signal. It results in no delay but requires many signal channels as well as appropriate signal preprocessing and a high amount of computation. The three methods are then discussed and compared in terms of filtering performance and real-time feasibility. We see that while ICA results in no delay, its execution time should be longer than filtering (which causes delays) due to its requiring the input sequence for its algorithm as well as preprocessing steps.
For the ML-based signal classification, three different approaches are tested. We start with a lazy method called k-nearest neighbors (k-NN) using the signal's root mean square value as a feature, which is effective but requires much computation effort. Deep reinforcement learning with a critic-actor network is then proposed (as an initial attempt), but this is shown to be not suitable for the situation due to its network not being compatible with time-series data. Lastly, a deep learning approach using a long short-term memory network combined with time-series feature extraction shows the best performance, given careful signal preprocessing and labeling. These methods are then compared in terms of accuracy and real-time feasibility. We shall see that to effectively work with time-series data, such as signals, the neural network needs to have a memory, which is the case of deep learning using a long short-term memory network that solves the vanishing gradient problem, and that the use of feature extraction can help solve the overfitting problem (making the training and test accuracies close to each other).
In the last part, the author analyzes the overall results and suggests some potential ways of further developing the topic.

1.2 About the research internship

1.2.1 The K. Nakano Laboratory and The University of Tokyo

The K. Nakano Laboratory is managed by Professor Kimihiko Nakano and is part of the Institute of Industrial Science at The University of Tokyo. It specializes in the control of mechanical and biological systems, combined into what we call human-oriented mobility engineering. Its core projects are shared control, human-machine interface (HMI), and high-level sensing, and controlling autonomous vehicles using sEMG signals is a topic of HMI.
For sEMG-related research, the laboratory has facilities such as sEMG electrodes, an armband, a laptop with a driving simulator, an accelerator, and a brake pedal for conducting experiments.
Copyright for using the following figure is obtained from the K. Nakano Laboratory, and it is related to [4] and [8].

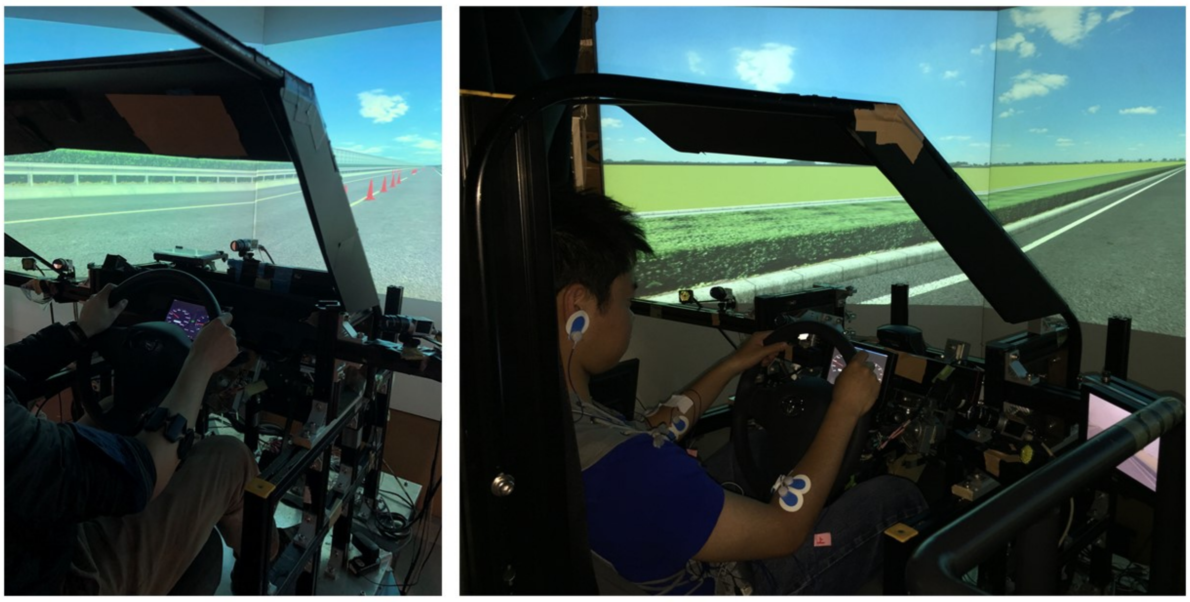


Figure 1: Facilities at the K. Nakano Laboratory for sEMG-based automotive control research.

Located in Tokyo, The University of Tokyo, abbreviated as Todai or UTokyo, is the first of Japan's imperial universities (now there are eight of them). Established in 1877, it is now one of this country's and the world's leading research institutions, and it receives the largest amount of funding from the government to carry out national research projects. It is one of the select Japanese universities assigned additional funding under the MEXT's Top Global University Project in an attempt to enhance Japan's global educational competitiveness.

1.2.2 Organization of the work

As the work is performed at a distance, the internship has to be organized according to the special situation. We have a team of researchers in Japan who go to the laboratory occasionally to conduct experiments with the test platform, collect the sEMG signals, and send them to the intern in France. The intern then works with these on his computer using MATLAB/Simulink and Python, and send the results/models to Japan. We organize online group meetings weekly to share results and discuss the next step. The efficiency of the work is certainly decreased in such a situation, but we are trying to achieve the best results possible. The author hopes to come to work on-site in the future.

2 Surface electromyography signal processing

This part presents and discusses the different signal processing strategies used to obtain a data set that would be used to train the ML agents. The related theoretical framework is found in Appendix A.

2.1 Signal measurement

In this work, sEMG electrodes attached to the driver's muscles generate signals corresponding to the movements of these and transmit their values wireless to the vehicle's computer, which will process the signals and use them for control. In Figure 1, we can see that the signals are measured using electrodes (blue and white) or an armband (black). These voltage signals are of amplitudes from a few microvolts to less than one millivolt and are sampled after each one millisecond.
In this part of signal processing, we would like to denoise signals obtained from the movements of five muscles in the driver's leg to control vehicle braking. These muscles are chosen based on the fact that they have large size and lie closer to the body surface, according to [9], thus making the measurement easier. These muscles and their respective positions in human anatomy are:

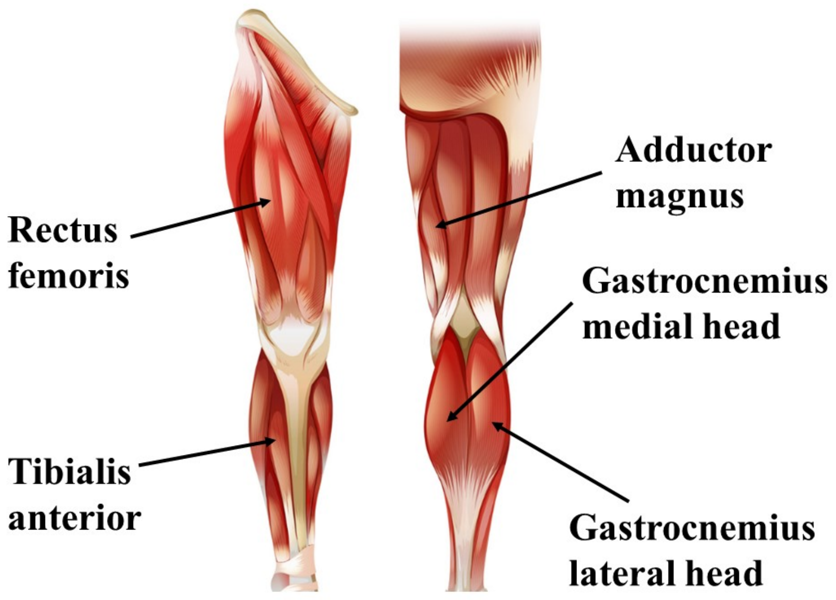


Figure 2: The five leg muscles whose activities we use for automotive control.

The first objective here is to analyze the different sources of noise through their frequency and power. Then we try to find a method that allows us to obtain the useful information contained in the sEMG while guaranteeing a sufficiently short execution time.
In a first attempt, we would like to develop tools to denoise the signals offline, i.e., take the whole signals and then process them within a certain time. We develop MATLAB script files for analyzing signals in the frequency domain and for analyzing the digital filters we design. We choose MATLAB as it can effectively work with double arrays and it has a toolbox for signal labeling, which is for data preparation. We test with ten sets of five signals for a total of fifty, saved in ten `.mat` files.
Here in the signal processing part, for brevity, the author presents only the work with one of the five muscles, which is the one corresponding to the activities of the tibialis anterior muscle. The other ones are processed similarly.

2.2 The different sources of noise contained in sEMG signals

To identify which sources of noise are in the acquired sEMG signals, we perform power spectral analysis to evaluate them in the frequency domain. In brief, such a method gives us the power that each frequency contained in the signal has, with a frequency range from 0 up to half of the sampling frequency. We plot the power spectral density (PSD) of the sEMG signals in dB.

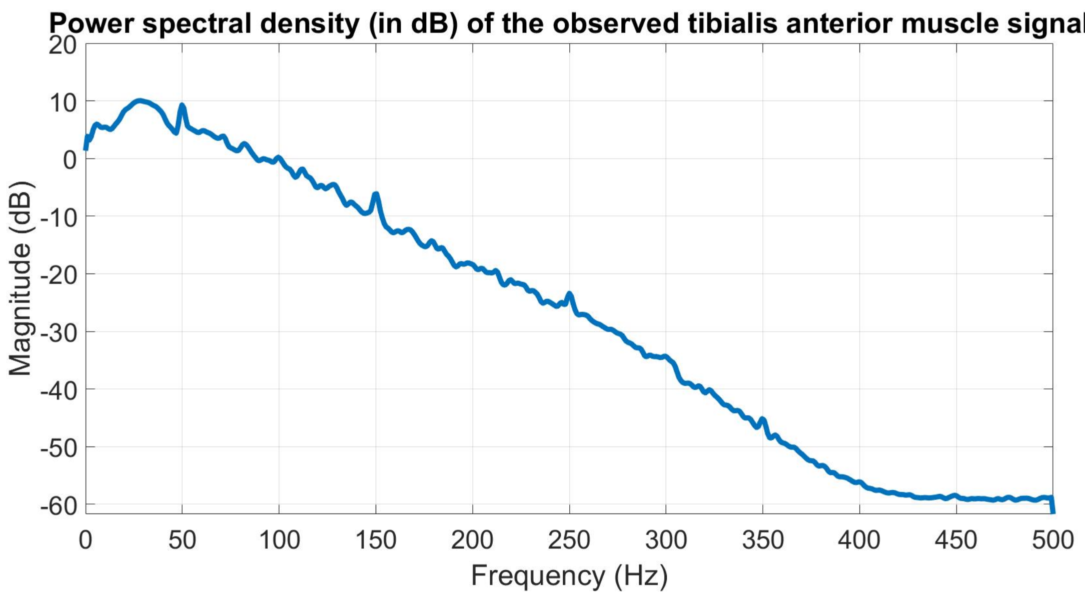


Figure 3: Power spectral analysis of an sEMG signal.

We see that there is a peak at 50 Hz and its odd harmonics. This is because the sEMG signals are contaminated by the noise from the Japanese electrical network whose frequency is 50 Hz. Also, the driver's heartbeat is a periodic noise, whose frequency ranges from 0.6–2 Hz, as the human pulse rate is generally between 40 and 120 beats per minute. The following figure shows an sEMG signal in the time domain.

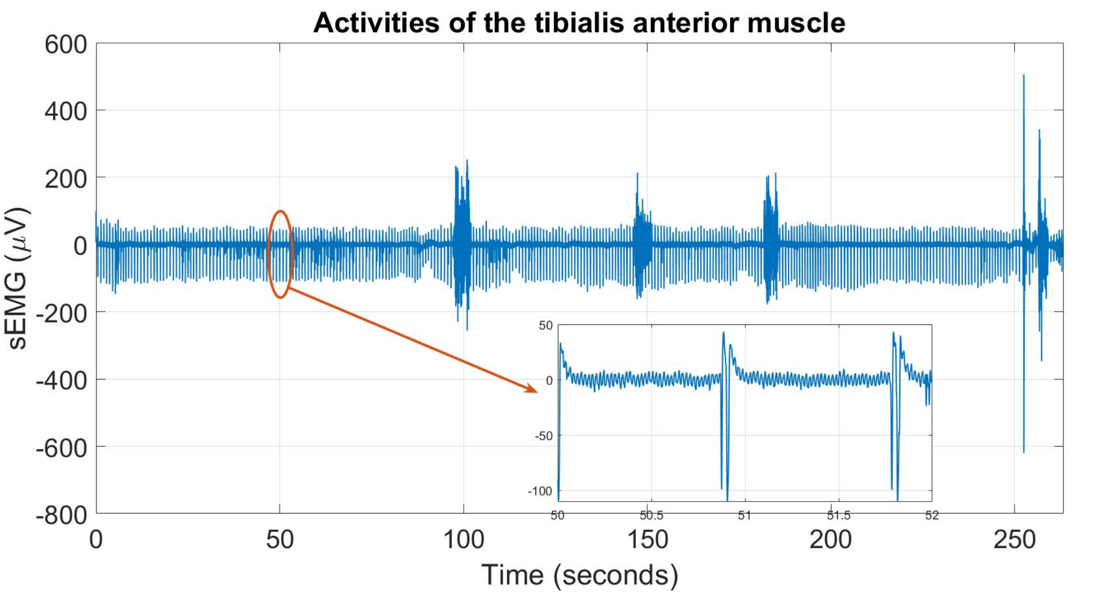


Figure 4: Analysis of the noise sources.

The figure in the time domain shows an example of a recorded sEMG signal, corresponding to the tibialis anterior muscle. The part containing the useful information can be seen in the background, covered by the noise. The greatest pulse at around 255s corresponds to the driver stepping on the pedal, and the smaller pulse at around 260s corresponds to releasing the pedal. The smaller oscillations at around 100s, 145s, and 180s correspond to the driver preparing for braking, but at these moments, the vehicle is not braked. We have similar remarks for the signals measured from the other four leg muscles, which are not shown here (for brevity).
The pulses we see at around 50.8s and 51.8s (after around one second) are caused by the heartbeat. As the heart is beating, the beat is transferred to the other body parts through blood vessels. It is worth noting that the weights and biases of artificial neural networks are highly sensitive to noise, and having clean data is, therefore, a crucial part of deep learning-based methods. Another source of noise is random noise coming from other muscles' activities or oscillations from the environment. Due to the randomness of such types of noise, it is advisable to measure sEMG signals from several muscles instead of only one, as we are doing. By doing so, we can neglect random noises and focus on filtering systematic noise.

2.3 Classical filtering

The first denoising approach is traditional bandstop filtering. The related theoretical framework is found in Appendix A.1. We make a bandstop FIR filter with a stopband from 49 Hz to 51 Hz to filter the electrical network noise, a transition band of 0.05, and analyze it in terms of poles – zeros, magnitude, and phase spectra.

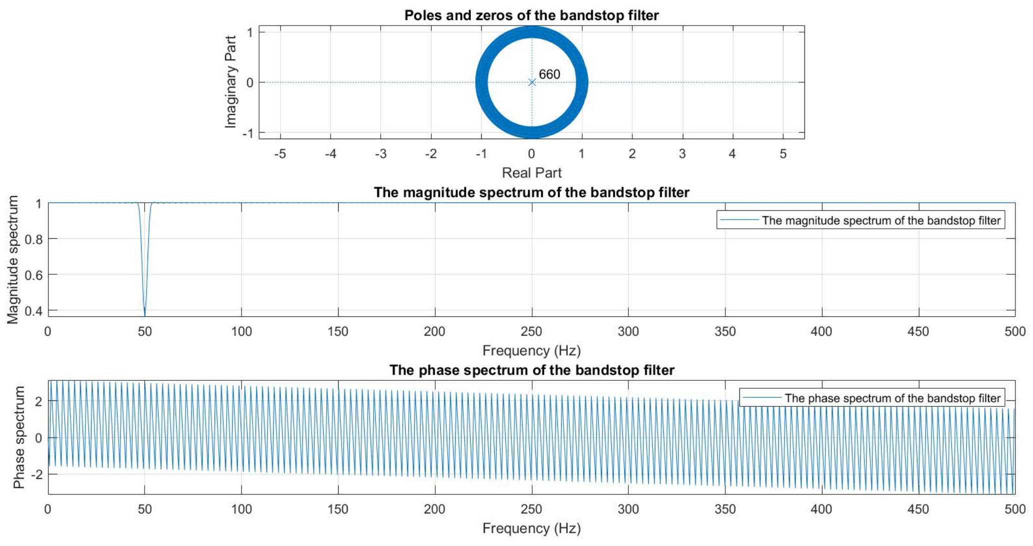


Figure 5: Analysis of the bandstop filter.

We see that this FIR filter is coherent with its theoretical background. Its order is 660, which is high. Its 660 poles lie at the origin, so it is stable, and it has a linear phase. Its magnitude spectrum confirms that it is a passive bandstop filter that cuts away the frequency 50 Hz. The following figure shows the results of applying the filter on five sEMG signals.

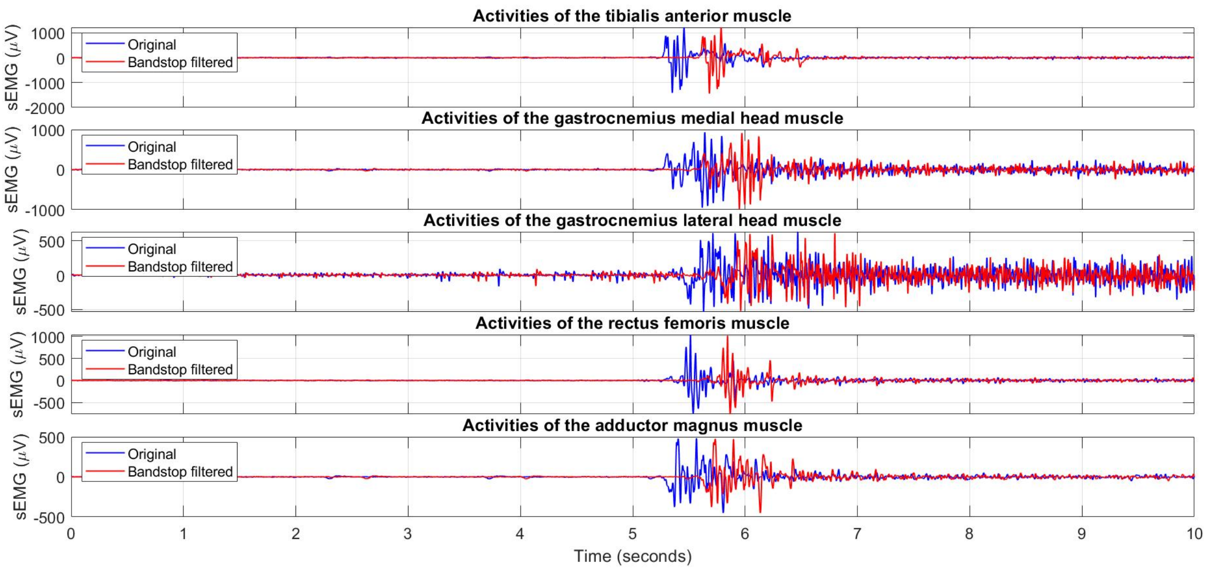


Figure 6: The results of bandstop filtering.

The problem with applying traditional filtering is that the signals would be delayed by a large number of samples. Keep in mind that the objective of this part is to control vehicle braking. In such a case, the response time of the algorithm is of high importance: we must be able to brake the vehicle without significant delays to guarantee driving safety. If we want to use a classical filter to denoise the signals, the filter's order would be 660 (for a transition band of 0.05). So the filtered signal would be delayed by that many samples. With a sampling period of 0.001s, the delay would be 0.66s. This means that a classical filter may not be suitable for real-time applications.

2.4 Optimal linear Wiener filtering

In this part, the author presents the application of an optimal linear filtering strategy called the finite impulse response (FIR) Wiener filter, which is more suited for real-time applications compared to its noncausal version. The related theoretical framework is found in Appendix A.2. It is worth noting that while this approach may seem complicated, it has some advantages compared to the traditional filter: it does not make us lose much useful information, and most importantly, it leads to a much smaller time delay of the processed signals. The procedure for applying this method is explained hereafter.
First, to determine an appropriate order for the filter, we need to look at the PSD of the signal and see that the width of the noise's frequency range Δf is approximately 4 Hz (from 48 to 52 Hz). The filter's order N is determined as:

$$\frac{3.1}{N} = \frac{\Delta f}{2 \times f_s} \tag{1}$$

With a sampling frequency of 1000 Hz, the order's filter is chosen as $N = 1500$. To determine the necessary delay, we perform an autocorrelation of the observed signal, and the amplitude starts becoming constant at around 0.035s (we need some trial and error for this).

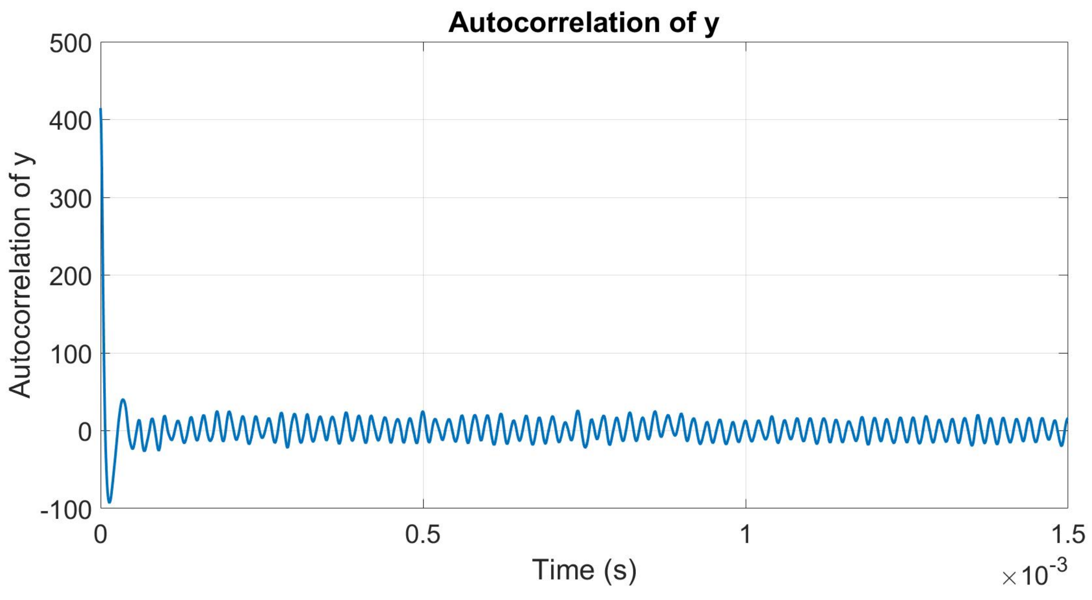


Figure 7: The autocorrelation of the observed signal.

Then we form and solve the Wiener-Hopf equation, whose solution is the filter, which gives us the estimated periodic noise. Subtracting the noise from the observed signal gives us a signal with much less noise.

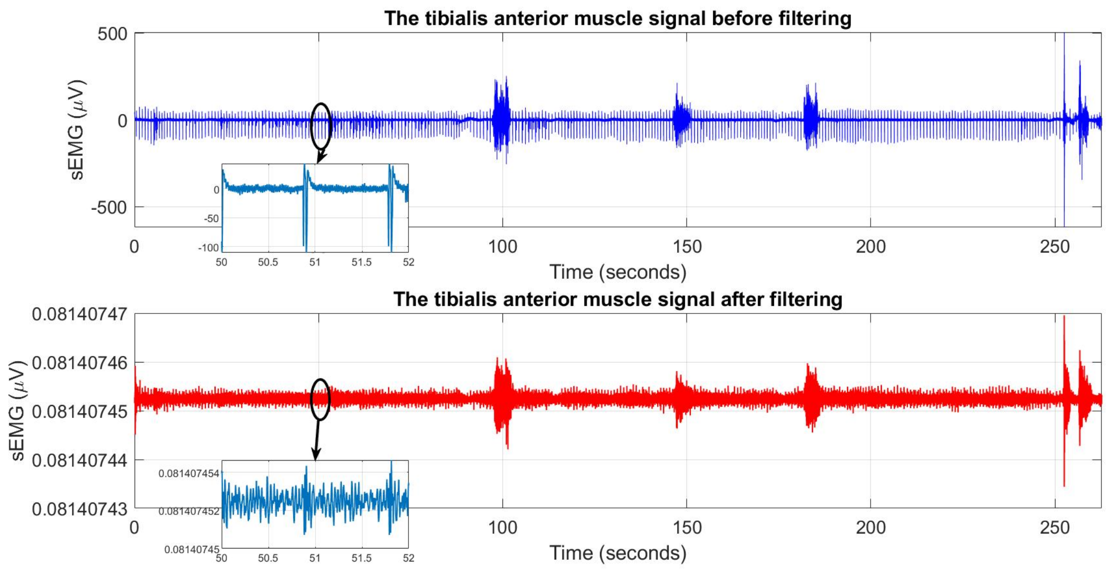


Figure 8: The results of Wiener filter.

This filter results in a much smaller delay compared to traditional filtering: only 0.035s. So it can be applied in real-time. Here are some remarks on this method. First, for each signal, there is an optimal delay, which we need to determine to filter it well. We do not need to know the exact value of this, but a value around 0.035s can be used. The fact that the filtered signal's amplitude is small leads to the need to rescale it. Moreover, different signals can have different amplitudes after they are filtered, so for methods such as deep learning, we should perform appropriate normalization when constructing the data set. These will be discussed more thoroughly in part 3.3.1.

2.5 Independent component analysis

The related theoretical framework is found in Appendix A.3. Theoretically, this method allows us to separate the independent components contained in our signals. We assume the signals are the different channels containing the useful information, noise from the electrical network, heartbeat, and from the other muscles, which are independent of each of the rest. As a first attempt, we hope to split the signals into two components: one would be the useful signals, and the other one is the combined noise.

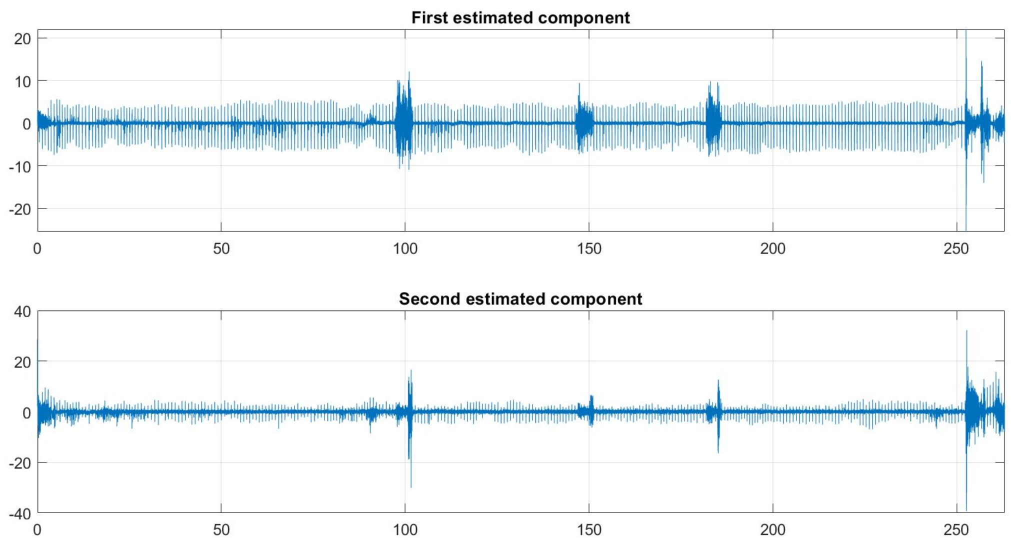


Figure 9: ICA with two components (too few).

It can be seen that the first component contains the noises, and the second component is the useful one. However, this model appears to be too simple for our case. We will consider the case where we separate the input into five different components, which is the highest number possible, as we have five signal channels.

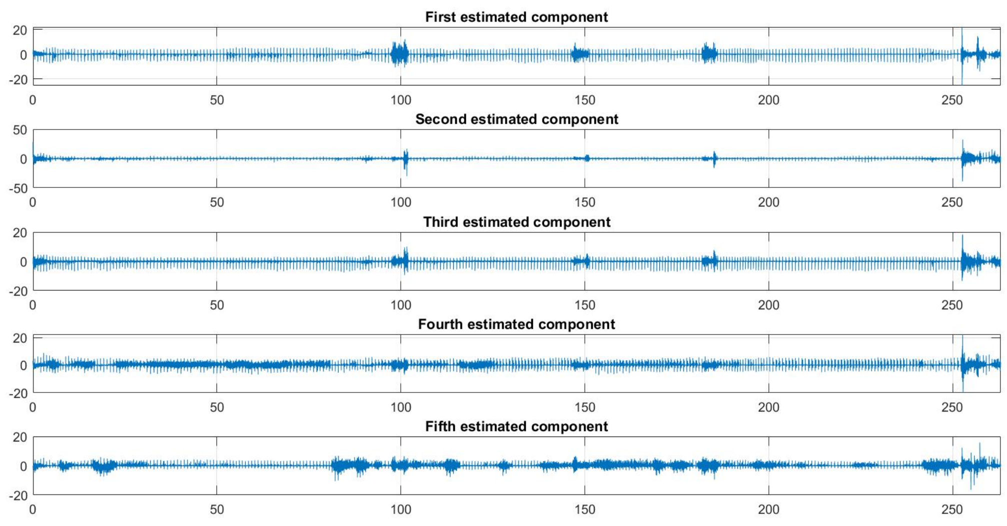


Figure 10: ICA with five components (too many).

This time, it can be observed that the second component is much cleaner than the other ones. However, having a higher number of independent components than necessary can lead to high computation effort required, which slows down the process. Note that the Fast ICA algorithm requires the observed signals to be executed, so in practice, we need to wait until the whole signal segment has been collected and that there are preprocessing steps for this method. Taking into account this trade-off, we need to balance between how well we need to process the data and how fast we want to do that. The figure below shows the results of ICA with four independent components.

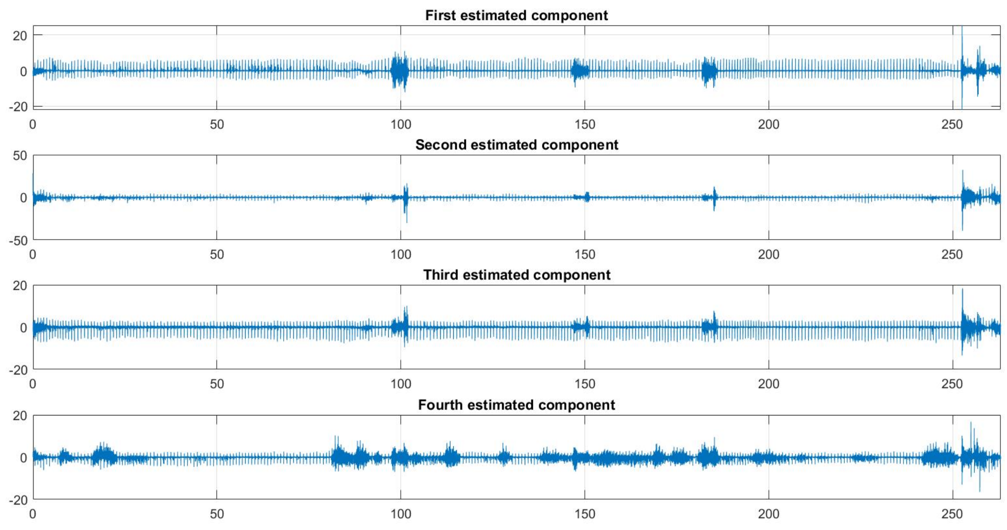


Figure 11: The results of ICA with four components.

It can be seen that the second components (the useful ones) of this case and of the previous one are not very different. This means that we can use ICA with four instead of five independent components and still get the useful signal. The model with five components is more complex than necessary.

2.6 Comparison of the proposed signal processing methods

Here we would like to compare the three proposed methods to see their advantages and disadvantages. It can be seen that traditional filtering is not suitable for real-time applications, as the delay in time is not negligible.

	Traditional filtering	Optimal linear filtering	Independent component analysis
Advantages	Simple, requires individual signals	Small delay, requires individual signals	No delay
Disadvantages	Large delay	Theoretically complicated	Requires all the signals, high computation effort, results not in order

Table 1: Comparison of the proposed signal processing methods.

Note that for the Wiener filter, we can calculate the filter's coefficients and store them, and when we collect the signals, we only need to apply the filter like in the case of traditional filtering (we assume the sEMG signals for each muscle stay the same in terms of nature). For the ICA method, we cannot do so, as its algorithm requires the observed signal. All the computation of ICA is performed when we collect the signal segment, which is quite similar to the k-NN ML method to be discussed later. Also, the ICA components are not in order, so we may need to establish conditions to check which one is the useful signal, which again causes a problem with real-time feasibility. On the other hand, the Wiener filter may result in the need to use different delays for different channels, so we can have problems if our ML method requires signals to be of the same length.

3 Surface electromyography-based automotive braking control

This part presents and discusses the different ML strategies used to classify the sEMG signals. The related theoretical framework is found in Appendix B.
Attention has to be paid to the execution time of each method, as the goal is to apply these in real-time while guaranteeing driving safety. The overall procedures to train as well as use the agents are presented in the following figure.

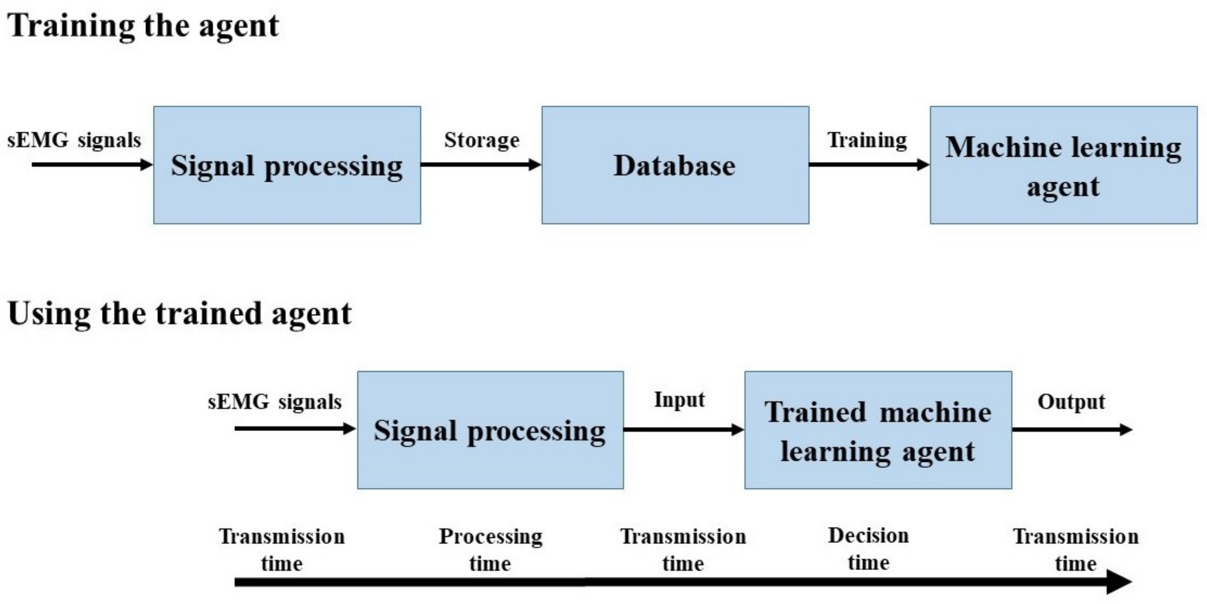


Figure 12: The procedures for training and using machine learning agents.

The amount of time needed to obtain the output includes the time it takes to process the input signals and the decision time, which depends on each of the algorithms. As we cannot reduce the transmission time, to reduce the total time, our objective here is to find a signal classification strategy that gives its results as fast as possible while guaranteeing accuracy (for driving safety).

3.1 k-NN classification

k-NN stands for k-nearest neighbor, a simple yet effective ML-based classification algorithm. The related theoretical framework is found in Appendix B.1. In this approach, we define some features of the data that allow them to be classified into classes, then a formula that gives the "distance" between data, according to these features. The test object will be classified using its k nearest data labels, according to this distance. Existing work concerning the use of k-NN to classify sEMG signals measured from arm muscle activities introduced six different features of sEMG signals [10].
In our application, we would like to take advantage of the root mean square (RMS) of the sEMG signals as a feature. This is because we can see that the signal part corresponding to vehicle braking has a much higher average value, and we use a single feature to reduce the execution time.
We will perform k-NN classification after each 0.1s, which means we take the RMS values of 100 signal samples. First, to build our data set, we cut the signals into segments of 100 samples each. We then give them labels of "braking" and "not braking", then store them in data sets for training and testing. Each time the k-NN algorithm is executed, a 0.1s signal segment (which contains 100 samples) has its RMS value compared to the RMS values of the whole training set, and the labels of the k nearest neighbors decide the label of the input.
Then we perform a loop across all the five signal channels with different values of k to determine the relationship between k and accuracy for each channel. For each of these, we perform k-NN with k varying from 1 to 20, use a loop to compare between the results and the ground truth, and plot the accuracy with respect to k. Note that the higher the value of k, the more computation is required.

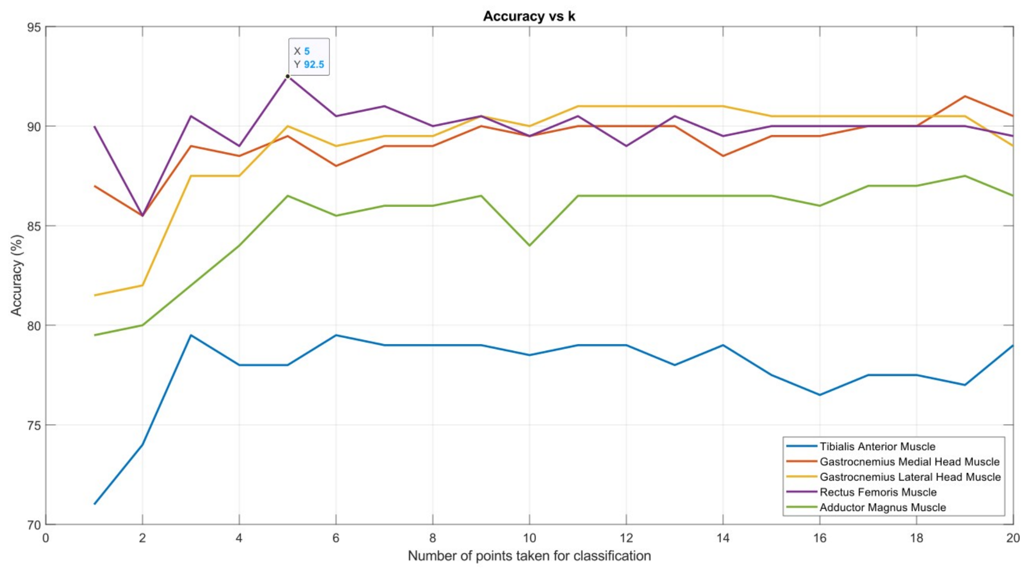


Figure 13: Looping to obtain the best value of k.

The result is that using 5-NN on the signal measured from the rectus femoris muscle can achieve the best accuracy of more than 92%. We then perform a simulation in which signal segments, each of 100 samples, along with their correct labels, are run one after another. We construct, from the test data set of the two labels, a test signal in which the segments of the two classes are run alternatively. We compare the label decided by the model with the ground truth to calculate the overall accuracy.

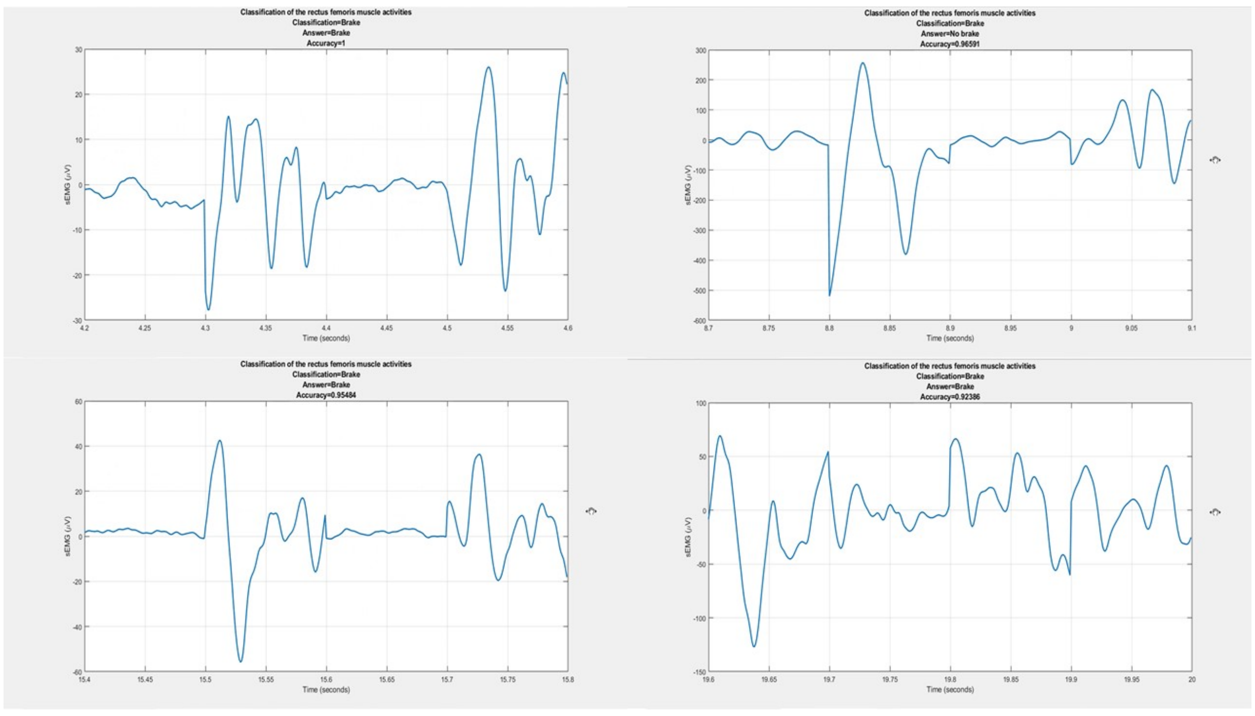


Figure 14: Applying k-NN to classify sEMG signals using real-time simulation.

The figure above contains the results cut from our video of the simulation. It can be seen that at first, the model classifies the signal segments correctly, so the accuracy is 100%. Then it starts to make some incorrect decisions, which lowers the overall accuracy, and finally, it ends up with an accuracy of 92.386%.
Such accuracy can be reasonable, given the simplicity of the method. However, we are trying to apply this method in real-time, so a lazy method such as k-NN may not be suitable due to its requiring much computation effort, which leads to a high execution time. We want to use pre-trained ML models that will respond fast enough to the input without too much calculation.

3.2 Deep reinforcement learning

This is our initial attempt to use deep reinforcement learning. In this method, an agent with a defined structure shall learn to classify the signals into two categories – braking and not braking the vehicle. The agent is trained according to the reinforcement learning framework, which is based on Q-learning. The related theoretical framework is found in Appendix B.2. Relevant studies from our laboratory [11] presented a comfort-oriented haptic guidance steering system via deep reinforcement learning, and we would like to deploy a similar network for finding the map from sEMG signals to the states of braking and not braking the vehicle. Here the rule we define the reward, and the DNN's structure are quite similar to those in the mentioned work.

3.2.1 Learning structure

In the framework of reinforcement learning, an agent interacts with a set up environment, performing an action on the environment, and measuring the outcome. This outcome is then evaluated by calculating the corresponding reward, which will lead to the agent being trained to maximize this reward by an algorithm of Q-learning.

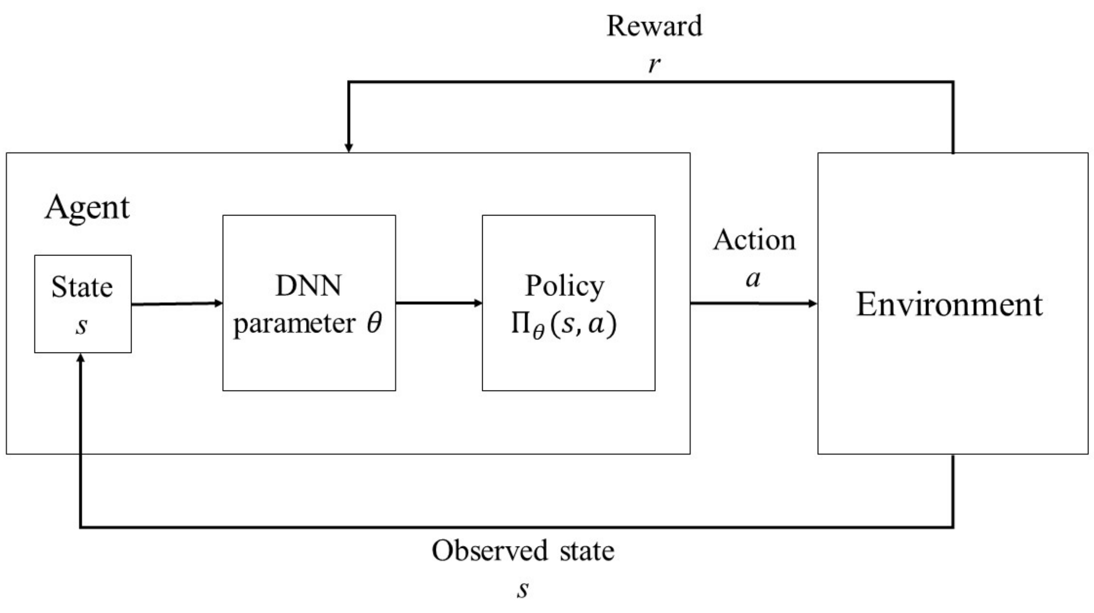


Figure 15: DRL's learning structure.

In this work, the agent is the vehicle's computer that takes the sEMG signals as the observed states, produces the decision signal (braking/not braking), and measures the reward defined to be as simple as:

$$r = -|pedalReference - decisionSignal| \tag{2}$$

To do this, we used a real pedal that produces an electrical pulse whenever it is pressed, serving as a reference signal. In testing and application, we do not observe this signal but only the sEMG ones. We implement a simple Simulink model quite similar to Figure 15 for this approach, which serves to simulate the interaction between the agent and the environment.

3.2.2 Agent structure

The agent has a deep neural network (DNN) structure, consisting of layers of neurons connected using activation functions. It consists of two networks: the critic and the actor.
The critic network serves to estimate the long-term reward given the states and actions. It consists of two smaller branches, called the state and action networks, of which the inputs are the observed state and action, respectively. Then it combines these two branches into one branch.
The actor network serves to determine the next action given the current state. It has one input, which is the state, and its output is the action.
The structures of the two networks used in this study are as follows:

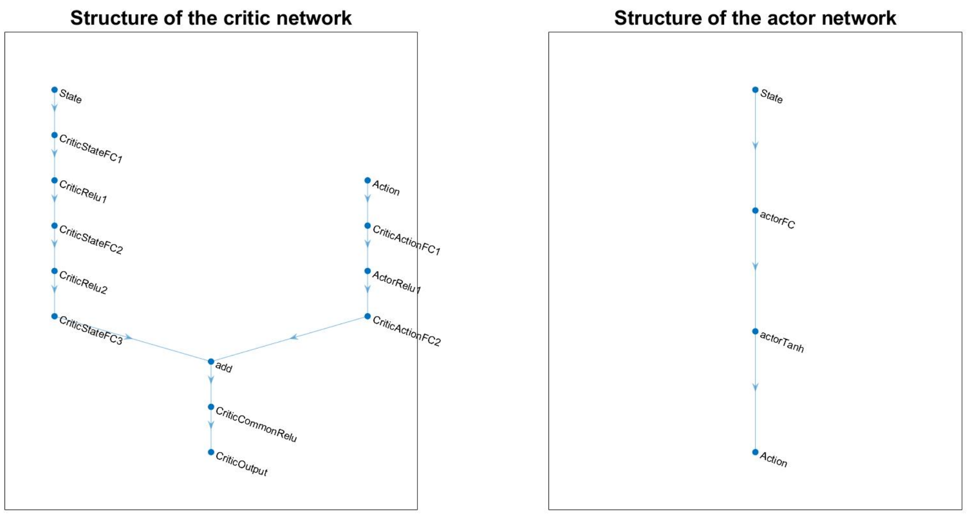


Figure 16: Structures of the critic and actor networks.

It has a classical structure. Then we train and test this agent using the observed signals as inputs and the pedal reference as the ground truth as below. The training algorithm is called deep deterministic policy gradient (DDPG).

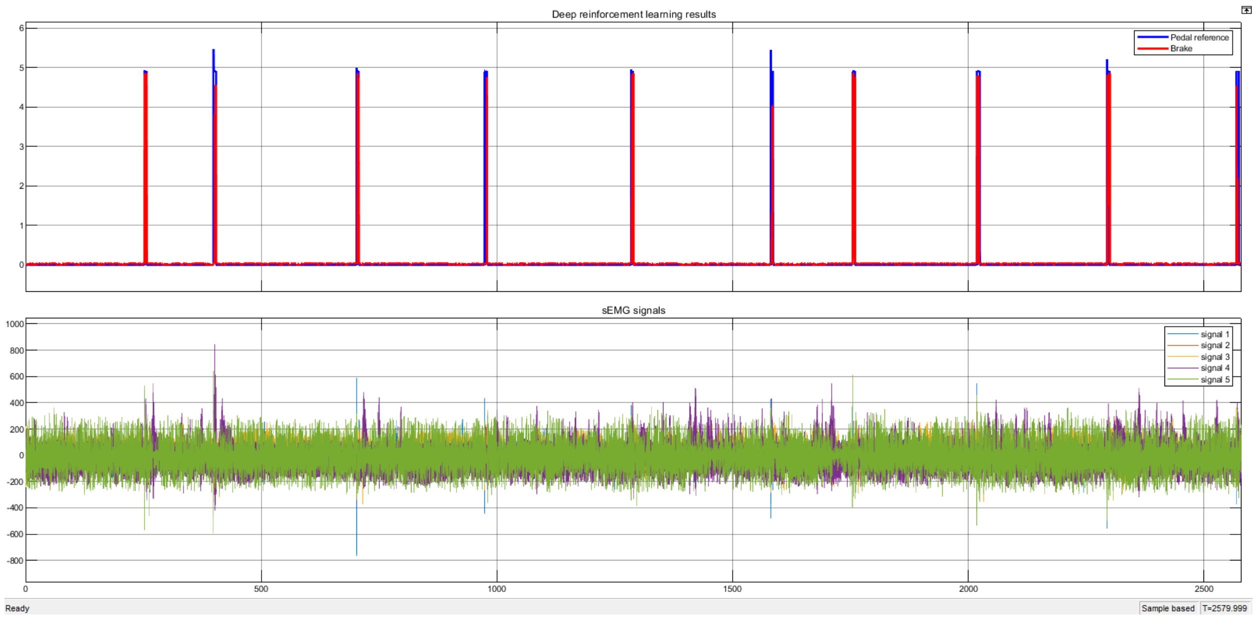


Figure 17: Applying DRL to classify sEMG signals.

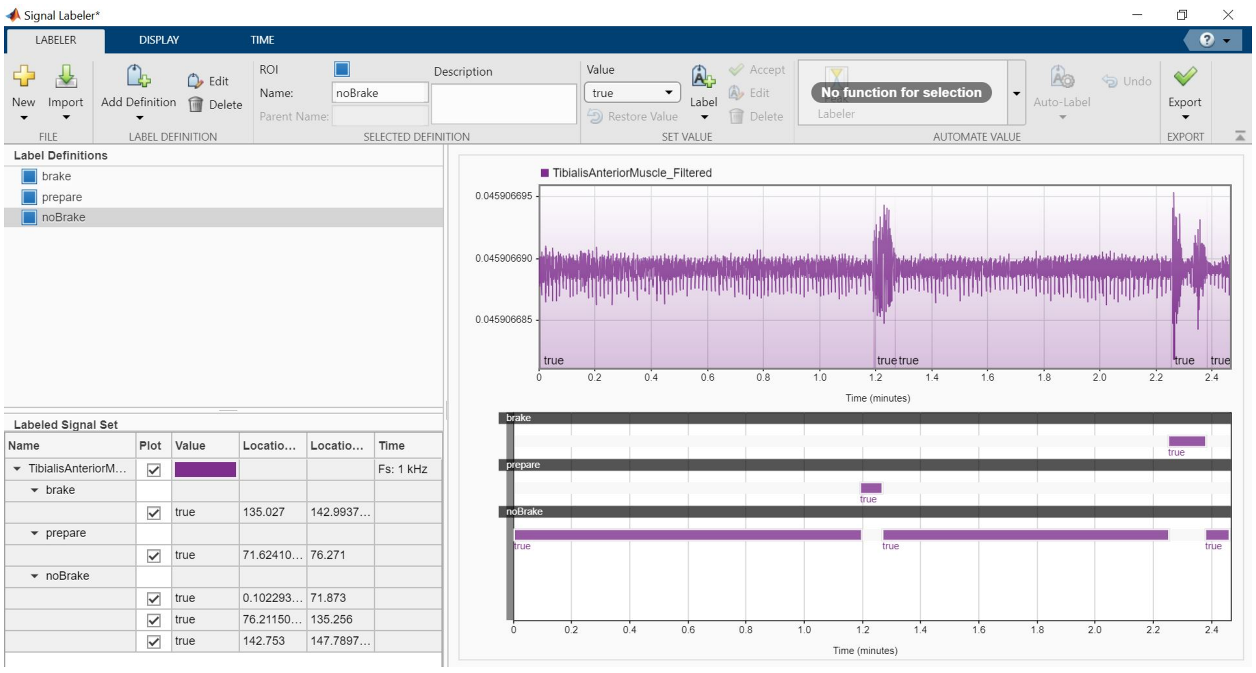
Our simulation has shown that if we take the reference from the pedal as an observation of the DRL structure, then the accuracy is guaranteed, as shown in the figure above. However, this approach does not work well if we observe only the five sEMG signal channels. In practice, we do not observe the pedal reference. We explain this by the fact that in our situation, the states from the environment to the agent are the next signal segments, which are independent of the existing states and of the action the agent takes. The DNN here is not suitable for time-series data, as it does not have a memory. So our situation may not be coherent with DRL.

3.3 Deep learning using a long short-term memory network

We propose a deep learning (DL) approach using a long short-term memory (LSTM) recurrent network, as this is well-suited to study sequence and time-series data. There has been work using LSTM to classify electrocardiogram signals [12]. However, they used a well preprocessed and labeled data set, and electrocardiogram signals are different from sEMG signals. This part presents first the methods used for processing and labeling sEMG signals, and then how to classify them using LSTM. The related theoretical framework is found in Appendix B.3.

3.3.1 Preparation of training and test data sets

To prepare a database for this method, we first need to display the filtered signals in the time domain to identify the regions of interest (ROIs). The tool is MATLAB Signal Labeler, which allows us to define labels and apply them directly to the signals. The three labels to use are “brake”, “prepare”, and “noBrake”. These labels have binary values, which means true and false.
As the filtered signals are not the same in length and shape, we have to label them by hand, which takes a fair amount of time and effort. Attention should be paid so as not to misplace the labels, which would lead to wrong decisions made by the network. The following figure demonstrates how a filtered sEMG signal from the tibialis anterior muscle is labeled. We need to repeat this for each of the muscles, for the data sets. In this work, as the filtered signals from the fifth and eighth data sets are not clean enough, we do not use them for this method in order not to lower the accuracy. We have eight labeled signal sets in total.



Labeled Signal Set

Name	Plot	Value	Locatio...	Locatio...	Time
TibialisAnteriorM...	✓	🟪			Fs: 1 kHz
brake					
	✓	true	135.027	142.9937...	
prepare					
	✓	true	71.62410...	76.271	
noBrake					
	✓	true	0.102293...	71.873	
	✓	true	76.21150...	135.256	
	✓	true	142.753	147.7897...	

Figure 18: Labeling signals’ regions of interest by hand.

Then, from these sets, we define the MATLAB cells containing the corresponding labels and use appropriate loops to obtain two MATLAB cells: one contains the signals (of different lengths), and the other one contains the labels. Here the “prepare” label is considered as not braking the vehicle.
It should be noted that different input sEMG signals are filtered differently, so the amplitudes of the filtered signals are not of the same range. We normalize this by the following steps:

- Subtract the mean of each signal from the signal segment. This is done after instead of before labeling to prevent the calculated mean from being affected by the other ROIs.
- Find the range (maximum – minimum) of each sample, then the amplitude (half of the range), then the average amplitude
- Multiply each of the segments by a coefficient depending on its amplitude with respect to the average amplitude

After this step, the signals of the same label can be scaled. Then, as the signals are not of equal length, we develop an algorithm to split these signals into as my smaller segments of equal length as possible and ignore the remainder. The reason to do this is that during training, the data are split into mini-batches. The function then pads or truncates signals in the same mini-batch, so they all have the same length. Too much padding or truncating can have a negative effect on the performance of the network because the network might interpret a signal incorrectly based on the added or removed information. Each signal segment contains 1000 samples, which corresponds to one second. The signals are split into as many segments of 1000 samples each as possible, and the remaining samples are ignored.
Now, we need to split the data set into a training and a test set. Because the data set is not

balanced between the two labels (we have more “noBrake” than “brake”), we would apply a method called oversampling to have the same number of signals for each label. If we do not do this, then the model would learn to put all the signals into the class with the most samples. The oversampling procedure is as follows:

- Split the signals according to their class
- Use the MATLAB `dividerand` function to divide targets from each class randomly into training and test sets
- Duplicate the data set with fewer samples to match the one with more samples, ignore some samples from the latter if necessary

In brief, the data preparation part for training an LSTM network consists of these steps:

- Filter the signals, to reduce the noise
- Label the signals in the time domain, according to the ROIs
- Split the signal segments into groups, according to the labels
- Normalize the segments in each group
- Split the segments (of unequal lengths) into as many small segments with equal length as possible
- Split the data set into a training set and a test set
- Perform oversampling to have an equal number of samples for all the labels in the training set and repeat this for the test set

3.3.2 Applying the long short-term memory network

The LSTM network used in this work has the following structure:

- A sequence input layer to receive the input signal (sequence)
- A bidirectional LSTM layer, to look at the time sequence in both forward and backward directions
- A fully connected layer, to learn high-level features in the data
- A softmax layer, which acts as an activation function
- A classification layer, to classify the input signals

The LSTM network’s structure is demonstrated in the following figure.

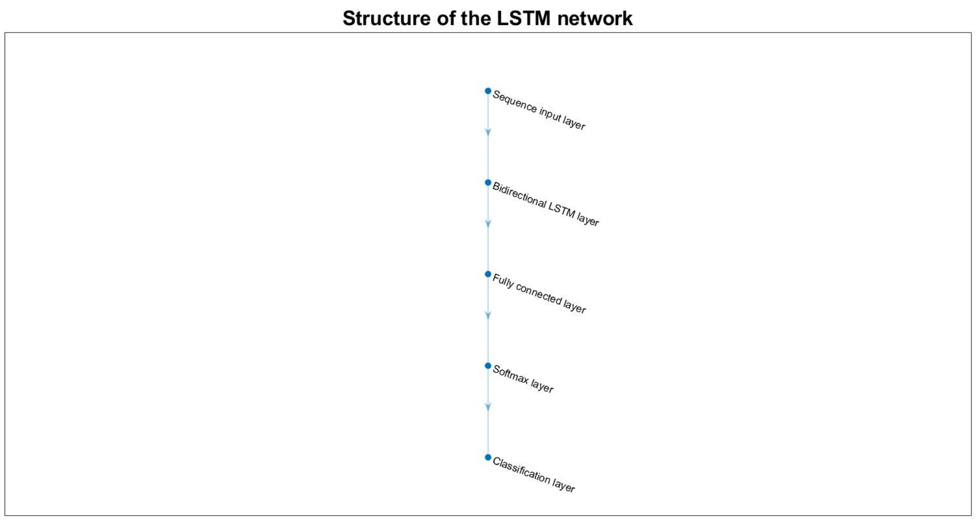
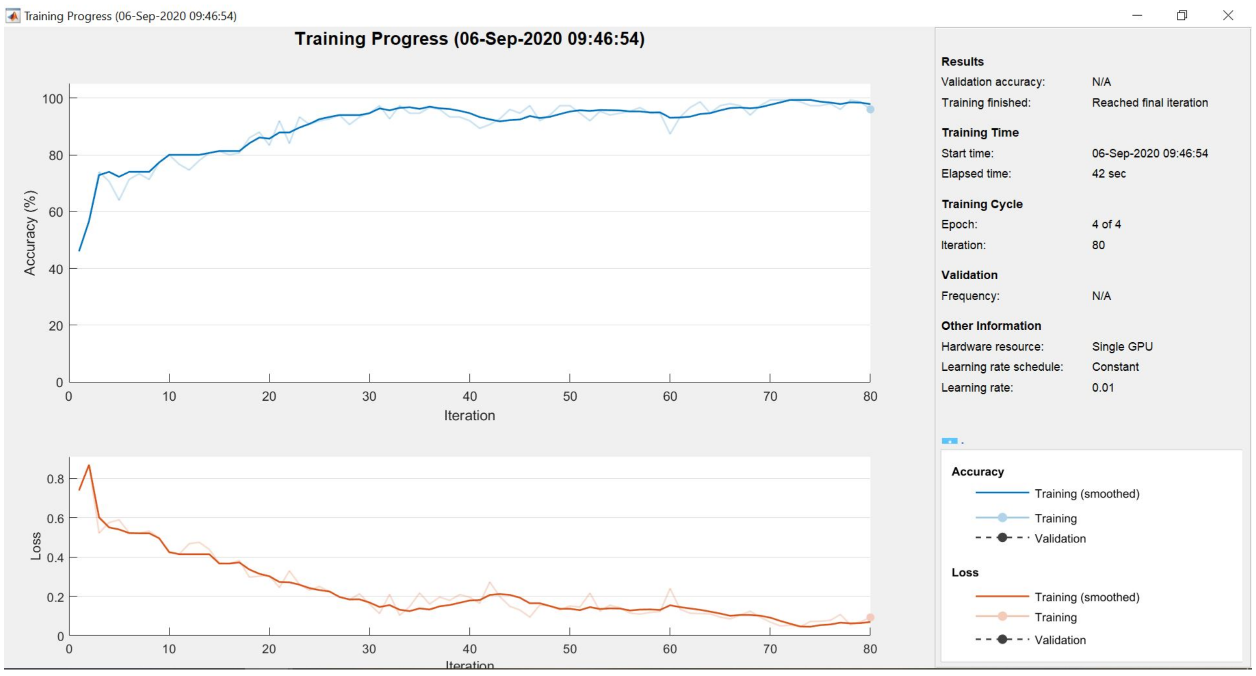


Figure 19: Structure of our LSTM network.

Here are some explanations on the bidirectional LSTM layer. Traditional DL with LSTM structure trains an LSTM network given the sequence input. This one uses two instead of one LSTM network. The first one learns the sequence input as-is, and the second learns the reversed sequence input. This way, the network is well suited for studying sequence input data, as it looks at the sequence in both directions.
We now train the network using the labeled data set prepared. Different training options are tried, with different numbers of epochs. Finally, we choose to train the network with four epochs, which leads to a quite small elapsed time of 42s (high training speed).



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 06-Sep-2020 09:46:54
Elapsed time: 42 sec

Training Cycle
Epoch: 4 of 4
Iteration: 80

Validation
Frequency: N/A

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.01

Figure 20: Training the LSTM network using signal segments as the input.

Then we test the network by letting it classify the signals in the test data set and form a test accuracy confusion matrix. Here are some remarks on how to read this. The target classes at the bottom are the ground truth, and the output classes on the left are the responses of the network. The true answers are put in the two green squares, and the two red squares contain the wrong answers. The overall accuracy is in the bottom right square.

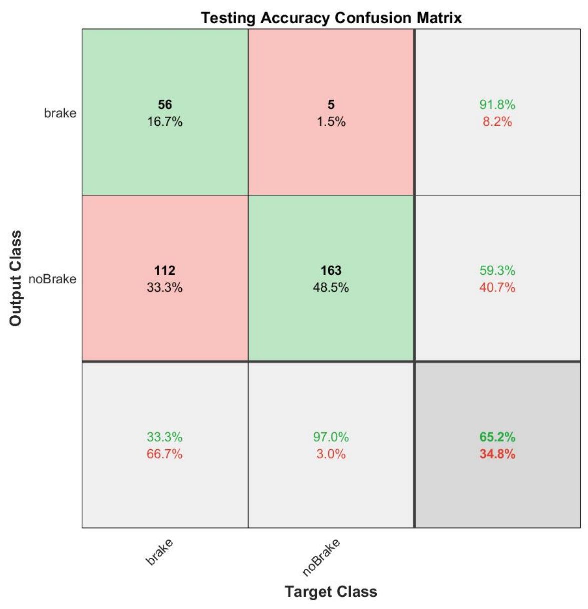


Figure 21: Test accuracy confusion matrix of LSTM using signal segments as the input.

The first attempt gives an overall training accuracy of more than 90%, but a test accuracy of only 65.2%, which is considered dangerous as we are working with vehicle braking control. When we have a high training accuracy but a low test accuracy, it is called overfitting, which means the model is so fit the training data that it does not fit the test data.

Besides the overall accuracy, we also need to look at the accuracy value at the bottom left square, as this corresponds to giving "brake" signals to the model. This one is closely related to driving safety, and here it is the lowest accuracy, which means driving safety is not guaranteed here.
In an attempt to increase the network's accuracy, we apply a method called feature extraction. Typically, in this approach, we compute time-series features, such as spectrograms, and use them for training the network.

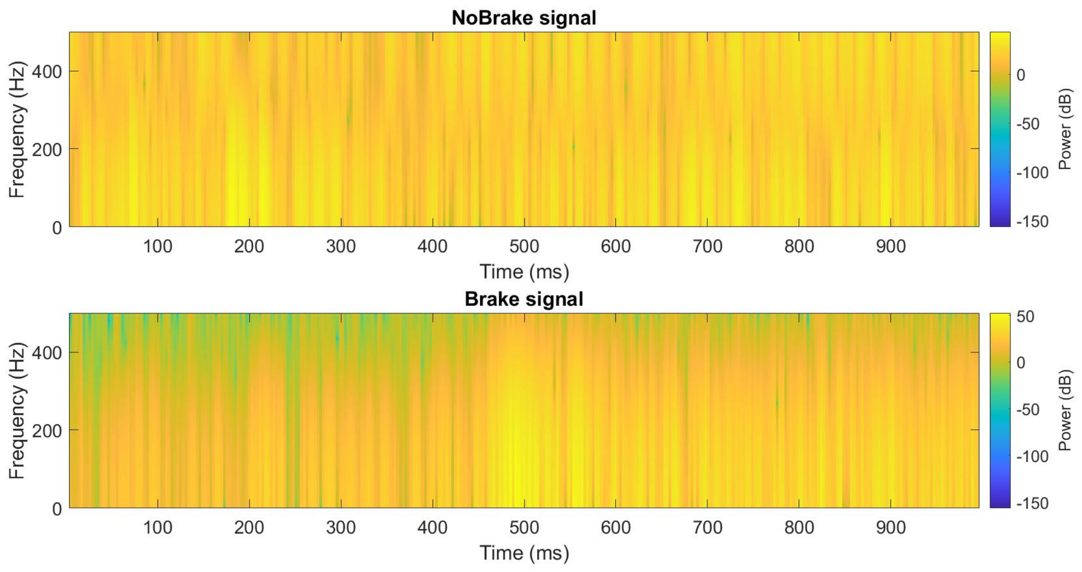


Figure 22: Spectrogram of the two types of sEMG signals.

As we use an LSTM network instead of a convolutional neural network, we need to adapt the approach to apply to one-dimensional signals. Time-frequency moments extract information from the spectrograms. Each moment can be used as a one-dimensional feature, which is an input for the LSTM network.
One of the moments in the time domain is the instantaneous frequency. This method using the MATLAB `instfreq` function estimates the time-dependent frequency of an sEMG signal (as the first moment of the power spectrogram). The function first computes a spectrogram using short-time Fourier transforms on the signal over time windows (here, it uses 255 time windows). The time outputs of the function correspond to the centers of the time windows.

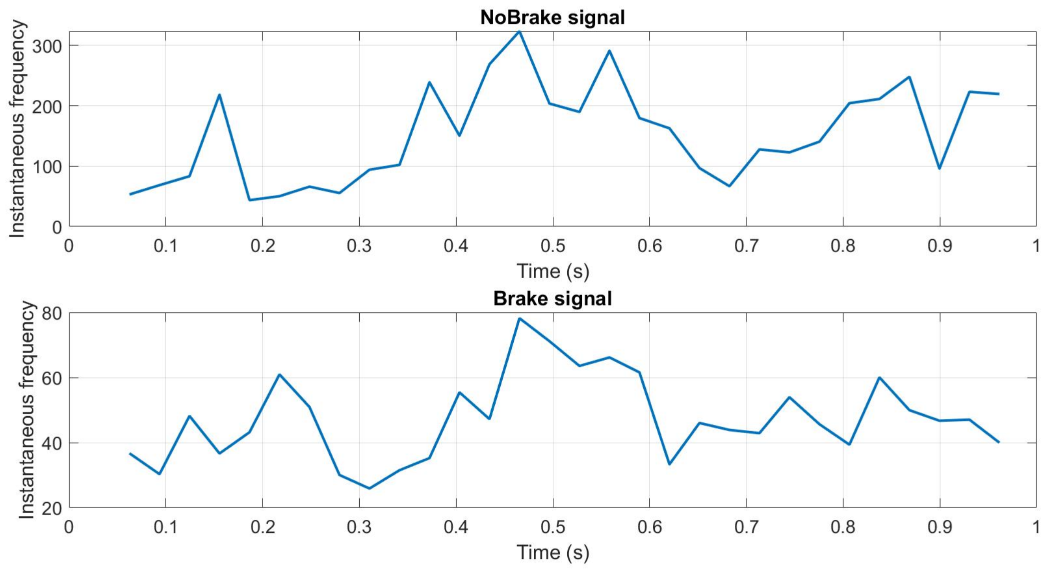


Figure 23: Time-dependent frequency plot of two types of sEMG signals.

We see that for sEMG signals corresponding to not braking the vehicle, the instantaneous frequency is much higher than that of the signals corresponding to vehicle braking (around 200 Hz compared to around 60 Hz). So, this time-series feature is excellent for signal classification.

This shows the importance of preprocessing the signals. If we do not filter the noise, then when we calculate the instantaneous frequency, we will have very similar values for the two classes. This is because they both contain much noise, and the frequency of the noise (50 Hz) will make it difficult to distinguish between the two cases.
The spectral entropy is another feature that measures how spiky/flat the spectrum of a signal is. The spectral entropy is low for signals with a spiky spectrum, e.g., a sum of sinusoids, and is high for signals with a flat spectrum, e.g., white noise. The MATLAB `pentropy` function estimates the spectral entropy of a signal from its power spectrogram, also using 255 time windows to compute the spectrogram. The time outputs of the function correspond to the center of the time windows.

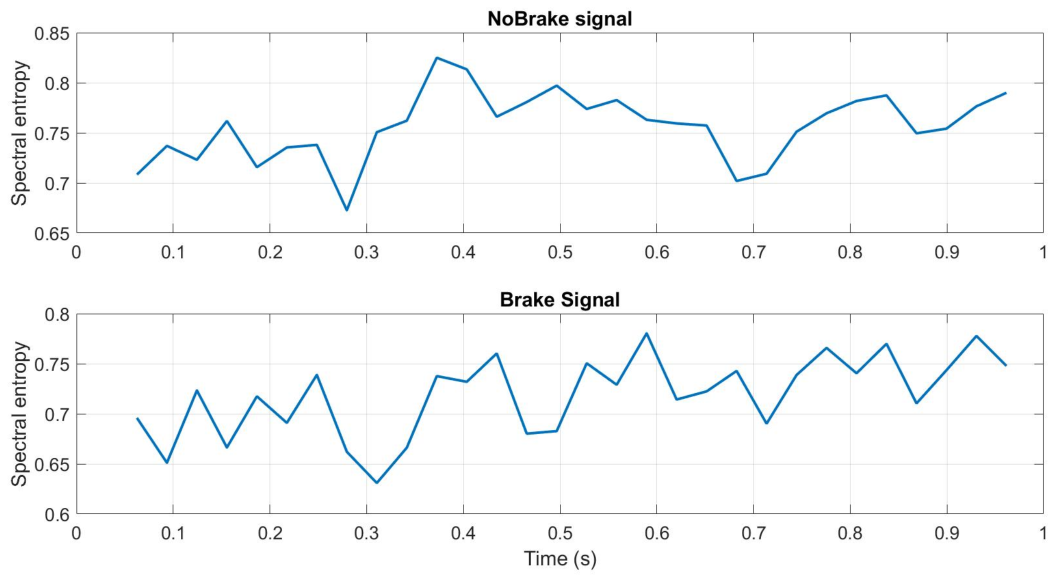
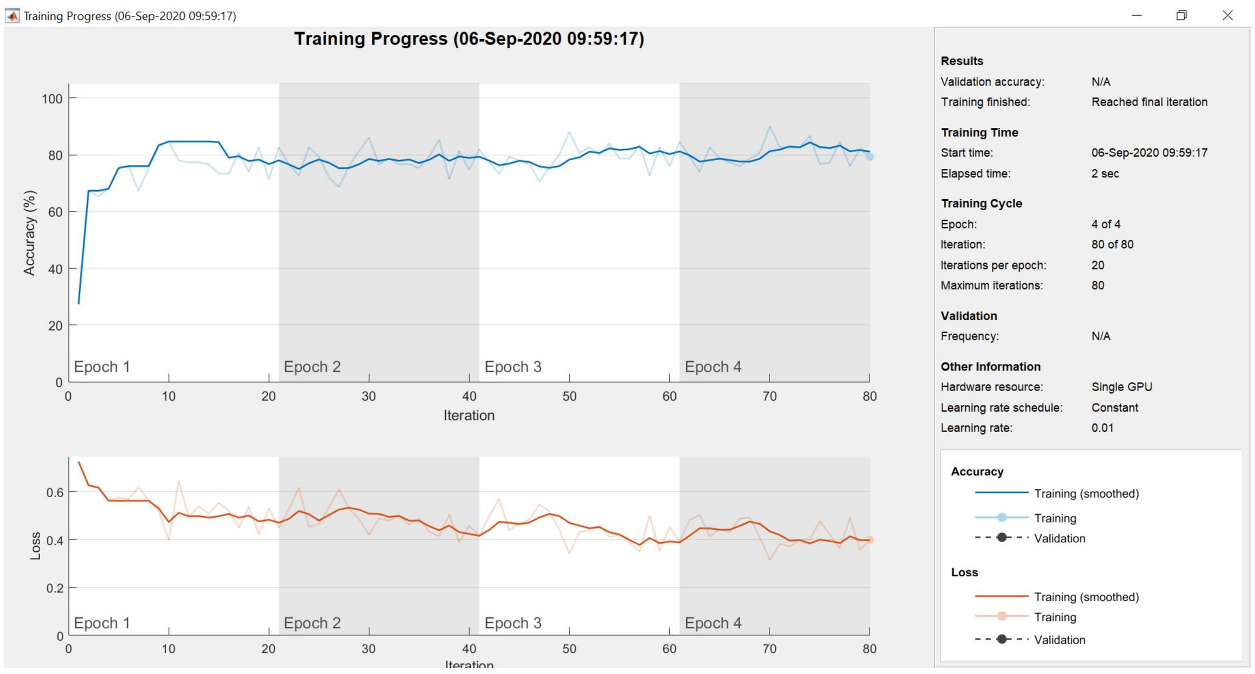


Figure 24: Spectral entropy of two types of sEMG signals.

We see that for sEMG signals corresponding to not braking the vehicle, the spectral entropy has a quite similar value to that of the signals corresponding to vehicle braking (around 0.75). However, they are quite different in the middle of the segment. So this time-series feature can be used for signal classification.
Now we need to standardize the data. The instantaneous frequency and the spectral entropy have different means, and the instantaneous frequency mean might be too high for the LSTM to learn effectively. If we train a network using data with a large mean and a large range of values, large inputs could slow down the convergence of the learning algorithm or the learning of the network [13]. To do this, we standardize the training and test sets using the training set mean and standard deviation.
The method here is standardization or z-scoring. Statistically, z-score is the number of standard deviations by which the value of a raw score (the observed value) differs from the mean of the measured data. This method includes subtracting the data mean from the raw score and then dividing the difference by the standard deviation of the data. In MATLAB, we use the `cellfun` function to apply the mentioned function on each cell of the cell array (where we store the extracted features). To return results in cell arrays, set the option `UniformOutput` to `false`. If this step is done correctly, then the mean values of the two features after the step must be much closer to each other than before. We repeat this standardization step for both the training and test data sets. In real-time applications, when we have new data, we need to standardize them using the available standardized data set before we can use them as the input of the network.
After the standardization step, a second attempt to use LSTM is then made. The network structure remains the same, while the number of inputs increases from one to two. We test with different numbers of epochs, and the optimal value is also four. Compared to the previous attempt, this time, the elapsed time is much shorter, considering the same number of epochs.



Results
Validation accuracy: N/A
Training finished: Reached final iteration

Training Time
Start time: 06-Sep-2020 09:59:17
Elapsed time: 2 sec

Training Cycle
Epoch: 4 of 4
Iteration: 80 of 80
Iterations per epoch: 20
Maximum iterations: 80

Validation
Frequency: N/A

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.01

Figure 25: Training the LSTM network using extracted time-series features as the input.

The training results in an accuracy of approximately 80%. We see that this time, the test accuracy is significantly improved to 82.4%. The overall accuracy is more or less affected by the initiation step of the learning process, where the weights are initiated. However, after some trials, our model has shown that the accuracy always varies around 80%. After all, the test accuracy cannot exceed the training accuracy. These two are close to each other, which means we have solved the overfitting problem. All the accuracies are over 80%, especially with the one in the bottom left square corresponding to the case we actually brake the vehicle being the highest.
This higher accuracy is because we use the extracted features as inputs for the model, and they allow for more accurate classification compared directly using the signal segments where the features are not clearly expressed. The network in the first case cannot learn the map between the segments and the labels, as this relation is not quite clear. Learning the map from the features to the labels would be easier for the network.
However, it is worth remembering that this also adds some other steps, namely calculating the features as well as their means and standard deviations, and standardization to obtain the input. So it would add much more computation to the system and thus may raise the problem of real-time feasibility, compared to the case we input the signal segments directly.

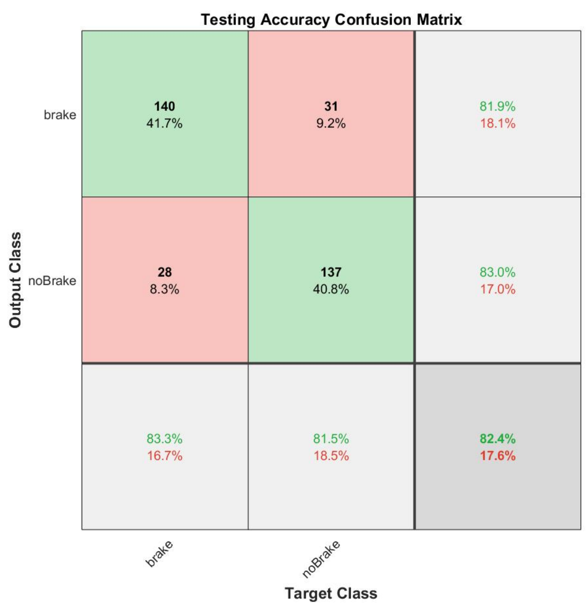


Figure 26: Test accuracy confusion matrix of LSTM using extracted time-series features as the input.

In brief, the steps for training an LSTM network with feature extraction are as follows:

- Construct a DNN of LSTM structure
- Select and form a data set of time-series features
- Standardize the data across different features, statistically
- Use the training data set to train the DNN, loop to find an optimal number of epochs
- Use the test data set to test the trained DNN

And the steps for applying an LSTM network with feature extraction to classify sEMG signals are as follows:

- Record the signal segment
- Denoise it by filtering/ICA
- Extract the time-series features
- Standardize the extracted features
- Use them as the input of the LSTM network

3.4 Comparison of the proposed signal classification methods

We would like to compare the three proposed signal classification methods, in terms of advantages and disadvantages, to evaluate their accuracy and real-time feasibility.

	k-NN	Deep reinforcement learning	Long short-term memory
Advantages	Simple, lazy method	Automatic learning	High accuracy
Disadvantages	Requires high computation effort	Has a structure not well suited for time-series data	Requires a well-preprocessed data set and feature extraction

Table 2: Comparison of the proposed signal classification methods.

It can be seen that given a good enough data set, the LSTM is the best method for real-time application. This is because it is well suited for time-series data such as signals, thanks to its having a memory. Also, bidirectional LSTM is considered better than traditional LSTM, as it studies the sequence input in both directions.
It has been shown in this work that the introduction of time-series features helps prevent overfitting. By adding features that are greatly different across the labels, we make it easier for the model to learn to distinguish between them. Besides, there is always a trade-off between high accuracy and a short execution time, as shown in the last method. Also, it is shown that data preprocessing is a crucial step in ML-based methods.

4 Conclusion

4.1 Result discussion

The results include the models of signal processing and machine learning, as well as the used codes, and some favorable experimental results. The method of preparing the labeled data set, extracting features, and training the bidirectional LSTM network leads to an accuracy of 82.4% when tested with the test data. This accuracy can still be improved, provided that we have cleaner sEMG signals. The intern also produces filtered sEMG signals and labeled data sets that can be used to train other models in the future.
In real-time applications, this method can allow for signal classification after each period of one second where the signal segment collected is filtered, and the extracted time-series features are fed to the trained LSTM network.
The best method would be DRL using an LSTM network with feature extraction. However, it is too advanced to be considered within the scope of this internship, especially when performed at a distance.
(2021 update) This work has resulted in the publication of the following paper: **Gia Quoc Bao Tran**, Zheng Wang, Yusuke Koge, and Kimihiko Nakano, "Surface Electromyography-controlled Automotive Braking Assistance System Using Deep Learning", 12th International Conference on Applied Human Factors and Ergonomics, New York, USA, July 2021.

4.2 Remarks

In this work, the research intern applied his knowledge in two major fields: signal processing and machine learning, to solve a real problem. This is the first time he has worked with machine learning, and he appreciates more chances like this. He believes this experience shall prepare him for his first official machine learning course, which will be in the next semester starting September 2020. He also got a chance to learn many signal analysis and statistics tools, which is essential for machine learning studies and research.
Besides, as the intern wishes to follow a research career, this internship equipped him with more experience and sharpened his research-related skills: finding and reading scientific documents, presenting results orally and in written form, and especially looking for funding opportunities.
Some important lessons:

- Machine learning methods rely heavily on a clean, well preprocessed, carefully labeled data set. The data preprocessing step is, therefore, a crucial part of the whole method.
- An approach may be effective when applied to offline problems. However, it is not feasible in real-time application due to its drawbacks: high computation effort required, unwanted delays, high complexity, etc.

4.3 Potential developments

The author suggests some ways to further develop this topic, as follows.

- Extend from classification to regression to control vehicle steering by calculating the steering angle from arm muscle activities
- Improve the deep reinforcement learning approach to apply it successfully: use the bidirectional LSTM network with feature extraction
- Apply the machine learning algorithms in real-time so that the agent can learn while the vehicle is traveling, maximizing adaptability

For the last suggestion, we can try using human ratings as feedback to constantly train the model, similar to [14], a type of supervised learning where the human user evaluates the accuracy of the model to train it from data.

A Theoretical framework of the signal processing methods

This part presents the theoretical background of the three applied signal processing methods. To avoid ambiguity, only the very necessary contents will be included and there will be no proof that can be found in other sources.

A.1 Classical filtering theoretical framework

This section is about classical signal filtering theories, in particular FIR filters. The response of such a filter has the form:

$$y[n] = \sum_{k=1}^{M} b_k x[n-k] \tag{3}$$

where $x[n]$ is the signal to be filtered, b_k are the filter's coefficients, and M is the filter's order. Compared to an infinite impulse response (IIR) filter, the FIR filter has the following advantages and disadvantages:

- It is always stable, which means the response is bounded if the input is bounded.
- It can have a linear phase, which allows for simpler calculations.
- But it may require too many coefficients compared to an IIR filter.

To design an FIR filter, we do as follows. First, we select the desired impulse response $h_d[n] = g_d[n - \frac{M}{2}]$ corresponding to the type of filter we wish to design, with $g[n]$ chosen according to the Table 3:

Filter type	$g[n]$
low-pass λ_0	$2\lambda_0 sinc(2\pi\lambda_0 n)$
high-pass λ_0	$\delta[n] - 2\lambda_0 sinc(2\pi\lambda_0 n)$
band-pass $[\lambda_1, \lambda_2]$	$2\lambda_2 sinc(2\pi\lambda_2 n) - 2\lambda_1 sinc(2\pi\lambda_1 n)$
band-cut $[\lambda_1, \lambda_2]$	$\delta[n] - (2\lambda_2 sinc(2\pi\lambda_2 n) - 2\lambda_1 sinc(2\pi\lambda_1 n))$

Table 3: Desired impulse response corresponding to the different types of filter.

where M is the filter's order, defined according to the transition band, and the filter is calculated according to the desired attenuation in dB as follows:

Window	Transition band $\Delta\lambda$	Attenuation (dB)	Mathematical expression $w_{M+1}[n], n = 0, ..., M$
Rectangular	$\frac{0.9}{M+1}$	-21	1
Hann	$\frac{3.1}{M+1}$	-44	$0.5 - 0.5cos\left(\frac{2\pi n}{M}\right)$
Hamming	$\frac{3.3}{M+1}$	-53	$0.54 - 0.46cos\left(\frac{2\pi n}{M}\right)$
Blackman	$\frac{5.5}{M+1}$	-74	$0.42 - 0.5cos\left(\frac{2\pi n}{M}\right) + 0.08cos\left(\frac{4\pi n}{M}\right)$

Table 4: Filter coefficients for different window types.

Finally, we calculate the filter's $M+1$ coefficients:

$$b_k = w_{M+1}[k] \times h_d[k], k = 0, ..., M \tag{4}$$

It can be observed that, as we calculate $y[n]$ from a delay $x[n]$, the first M samples of $y[n]$ will be 0 (suppose $x[n] = 0, n < 0$), and then we start from the next sample. This means that the filtered signal will be delayed by a number of samples equal to the filter's order.
In Matlab, we apply the same procedure. However, when we calculate the filter's coefficients using the `fir1` command, the frequency to give to this command is $\frac{2 \times f_{cut}}{f_s}$, where f_s is the sampling frequency. Then to filter the signals we use the `filter` command.

A.2 Optimal linear filtering theoretical framework

This part presents the background theories of optimal linear filtering, in particular, the FIR Wiener filter which is used to process periodic noises. This filter was proposed in [15], while its discrete-time equivalence was proposed independently in [16]. We consider the following model in which the observed signal is considered to be composed of two components:

$$y(t) = x(t) + v_p(t) \tag{5}$$

where $y(t)$ is the observed signal, $x(t)$ is the useful signal (centered, large-band, and stationary up to second order), and $v_p(t)$ is a periodic noise uncorrelated with $x(t)$. The Wiener approach is to estimate this periodic noise $s(t) \equiv v_p(t)$ and then the estimated useful signal is calculated from $x(t) = y(t) - v_p(t)$. The filter's impulse response w is the solution of the Wiener-Hopf equation:

$$\Gamma_{sy} = \Gamma_y w \tag{6}$$

Which means:

$$\begin{pmatrix} \Gamma_{sy}[0] \\ \Gamma_{sy}[1] \\ ... \\ \Gamma_{sy}[N] \end{pmatrix} = \begin{pmatrix} \Gamma_y[0] & \Gamma_y[1] & ... & \Gamma_y[N] \\ \Gamma_y[1] & \Gamma_y[0] & ... & \Gamma_y[N-1] \\ ... & ... & ... & ... \\ \Gamma_y[N] & \Gamma_y[N-1] & ... & \Gamma_y[0] \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ ... \\ w_N \end{pmatrix} \tag{7}$$

The matrix Γ_{sy} is obtained from the cross-correlation between $s(t)$ and $y(t)$ and Γ_y is obtained from the autocorrelation of $y(t)$.
Because all we know is $y(t)$, we estimate Γ_{sy} to be equal to Γ_{yy_d} where $y_d(t) = y(t - \tau')$ with a large enough delay τ' (a large enough number of samples). This delay is determined as follows. We obtain the autocorrelation of $y(t)$, then we find the time t_c from which the amplitude of this autocorrelation remains unchanged, and $\tau' = t_c \times f_s$ where f_s is the sampling frequency. This t_c is then the delay in the time domain for the filtered signal which normally takes a very small value. For brevity, the proof is not shown here.
The procedure for applying this filter is as follows:

- Obtain the observed signal's PSD to detect the noises and the width of the peaks corresponding to them and then estimate the filter's order
- Perform autocorrelation of $y(t)$ to estimate the delay
- Perform cross-correlation between $y(t)$ and $y_d(t)$
- Form the Wiener-Hopf equation and solve it to obtain the filter's impulse response
- Filter the observed signal to have the estimated periodic noise
- Subtract this estimated noise from the observed signal to obtain the estimated useful signal

Note that this type of filtering does not result in a long delay in signals like the case of a classical filter, so it can be applied more safely in our case. Another advantage, compared to ICA, is that we only need the individual signals for this method, instead of all the signals as in the case of ICA. This can simplify the calculations: in application, this is as simple as a classical filter.
In MATLAB, we use the `xcorr` command to find the cross-correlation and autocorrelation of signals. The mode to be selected is `unbiased`. To construct the matrix Γ_y from the autocorrelation of $y(t)$, we use the command `toeplitz`. Then if we use `filter` on the delayed signal, we will get the estimated periodic noise.

A.3 Independent component analysis theoretical framework

Independent component analysis (ICA) is a widely-used blind source separation technique, which has many applications in image and signal processing.
In this method, we assume that we measure M linear mixtures x_m of N independent components s_n. In vector form, let $x = (x_1, ..., x_M)^T$ and $s = (x_1, ..., s_N)^T$ be the vectors containing the components. Note that each component is a vector in the time domain. We can assume

that both the mixture variables and the independent components have zero mean: in case they do not, then we can always subtract the mean from the observed components. We have:

$$x = As \tag{8}$$

where A is an $M \times N$ matrix characterizing the linear combinations. The two assumptions for ICA are that the source signals are independent of each other and that the values in each source signal have non-Gaussian distributions. The goal here is to estimate A, in other words, separate the independent components: from A, we find its inverse W and then:

$$s = Wx \tag{9}$$

This is done by adaptively calculating the w vectors and setting up a cost function which either maximizes the non-gaussianity of the calculated $s_n = w^T x$ (kurtosis and negentropy) or minimizes the mutual information.
Some ambiguities of ICA include:

- The extracted independent components are not in order.
- Changing the independent components' signs does not affect the ICA model.

Before performing ICA, it is advisable to preprocess the data using the following methods. The first one is centering: we subtract the input's mean from it, so the observed signals have zero mean. The mean will be stored to later add back to the estimated source signals. The second method is whitening, which means we transform the signal in such a way that the potential correlations between its other components are removed and the variance of each component is equal to one, which indicates the independence among the components. On the other hand, it simplifies so that the covariance matrix of the whitened signal is equal to the identity matrix. Whitening a signal involves the eigenvalue decomposition of its covariance matrix. The corresponding mathematical equation is [17]:

$$\tilde{x} = ED^{-1/2}E^T x \tag{10}$$

Where D is a diagonal matrix of eigenvalues, E is an orthogonal matrix of eigenvectors and $\tilde{x}$ is the whitened signal.
After these pre-processing steps, we use an algorithm called Fast ICA to update and find the weights in the W matrix to retrieve the independent components. In this algorithm, we need to use the equations g and g' which are derivatives of $f(u)$ as defined below.

$$\begin{aligned} f(u) &= log(cosh(u)) \\ g(u) &= tanh(u) \\ g'(u) &= 1 - tanh^2(u) \end{aligned} \tag{11}$$

The Fast ICA algorithm is presented below [18].

Algorithm 1: The Fast ICA algorithm

```
Result: The matrix $W$ to extract the $C$ independent components from the
        whitened input $X$
for $p$ = 1, $C$ do
    Randomly initialze $w_p$
    while $|w_p^T w_{p-1}^T - 1| \geq threshold$ do
        $w_p = \frac{1}{M}(Xg(w_p^T X) - g'(w_p^T X)w_p)$
        $w_p = w_p - \sum_{j=1}^{p-1}(w_p^T w_j)w_j$
        $w_p = \frac{w_p}{\|w_p\|}$
    end
end
$W = [w_1, ..., w_p, ..., w_C]$
```

Then we apply this W matrix onto the mixtures to obtain the independent components.

B Theoretical framework of the machine learning methods

This part presents the theoretical background of the three applied machine learning methods. To avoid ambiguity, only the very necessary contents will be included, and there will be no proof that can be found in other documents.

B.1 k-NN classification theoretical framework

The k-nearest neighbors algorithm (k-NN) is a non-parametric machine learning method used for both classification and regression, where the input consists of the k closest training examples in the feature space. In k-NN classification, an object's class is determined by doing a plurality vote of its neighbor objects: the object is assigned to the class most common among its k nearest neighbor objects.
Here k is a positive integer, typically small, that we need to determine to achieve the best performance while avoiding too much computation. If k has a too low value, then the accuracy would be lowered as the decision is based on too few nearest examples. However, if k is too large, then the computation would take more time to execute and can therefore raise a problem of real-time feasibility. For binary classification problems (in which there are two classes), such as determining whether there is a pedal brake or not, it is advisable to use an odd value for k to avoid tie votes. We can perform loops with different values of k to find the one that gives the best accuracy while not being too large.
This algorithm is a type of instance-based (lazy) learning, which means all computation is only performed at the time of function evaluation. Normalizing the training data can improve the method's accuracy significantly, as it relies on distance for classification. Distances can be defined using p-norms such as:

$$d(a,b) = \|a-b\|_p = \left(\sum_{i=1}^{n} |a_i - b_i|^p\right)^{\frac{1}{p}} \tag{12}$$

To further enhance accuracy, we can assign certain weights w_i to the k neighbors. This is called weighted k-NN. First, we calculate the distances d_i from the input to all the objects of the data set, then the input's label is determined by the weighted sum of the labels $label_i$ of the k nearest neighbors.

$$label = \sum_{i=1}^{k} w_i(d_i) \times label_i \tag{13}$$

where the weights follow a defined function of the distance such that the closer the object to the input, the higher the value of the weight, and they add up to one for normalization.

B.2 Deep reinforcement learning theoretical framework

This method utilizes the reinforcement learning structure to train a DNN agent for specific tasks. About reinforcement learning, it is modeled as a Markov decision process. An agent whose set of actions is characterized by A interacts with an environment in discrete time steps, resulting in agent and environment states that are then evaluated through an immediate reward rule.
At each time t, the agent receives the current state s_t and reward r_t. It then chooses an action a_t from the set of available actions, which is subsequently sent to the environment. How the action is chosen depends on the trained agent. The environment then moves to a new state s_{t+1} which depends on its nature and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined. The objective is to learn a policy that determines the action given the states to maximize the reward.
The agent of DRL follows a DNN structure, which consists of a critic and an actor network. For the critic network: it has two inputs (observation and action) and one output (critic). For the actor network: it has one input (observation) and one output (action). The critic network's role is to estimate the long-term reward used for the Q-learning algorithm, while the actor network is to decide which action the agent should take given the current observed states and reward.

The agent learns its tasks through the deep deterministic policy gradient (DDPG) algorithm. DDPG uses four neural networks:

- A Q network θ^Q
- A deterministic policy network θ^μ
- A target Q network $\theta^{Q'}$
- A target policy network $\theta^{\mu'}$

The DDPG algorithm is presented below [19].

Algorithm 2: DDPG algorithm

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and
 $\theta^\mu$
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for $episode = 1, C$ do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for $t = 1, T$ do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and
         exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe the new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{Q'})\theta^{Q'})$
        Update critic by minimizing the loss:
            $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^\mu))^2$
        Update the actor policy using the sampled policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end
end
```

This training algorithm is described as follows:

- Experience replay: we use a replay buffer to sample experience to update the neural network parameters.
- Actor (policy) and critic (value) network updates: as is done in Q-learning. The updated Q value is obtained by the Bellman equation. However, in DDPG, the next-state Q values are calculated with the target value network and target policy network.
- Target network updates: we make a copy of the target network parameters and have them slowly track those of the learned networks via "soft updates".
- Exploration: this is done via probabilistically selecting a random action.

Here the replay buffer R is a large table where we store experience data (called transitions) from the past experiments in the form (s_i, a_i, r_i, s_{i+1}) which corresponds to [state, action, reward, next state] so that batches of these can be sampled later to train the networks.

B.3 Deep learning using a long short-term memory network theoretical framework

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture.
In traditional neural models, the inputs are considered as independent data. If, instead, the inputs are time-dependent sequences, then these models do not work well. To solve this problem, humans developed the RNN, which considers the input to be a sequence of data, using a type of memory to save past information to make better decisions. RNN uses only one neural network to obtain the output of each timestep. The outputs, when becoming inputs, are multiplied with a weight matrix. The results given by RNN at each timestep is related to the previous timestep.
Normally, using RNN is effective only if the input size is not very large (long-term dependency). If it is, then it will raise the vanishing gradient problem. An RNN uses a gradient to update the weight matrix, and its value decreases along with the layers. With a large input, the gradient will become too small for the network to keep learning. The RNN thus cannot store the first input information with large inputs.
Designed to avoid long-term dependency, unlike standard feedforward neural networks, LSTM has feedback connections. It can process entire sequences of data (such as a signal), so LSTM networks are suitable for classifying time-series data. The most important point is that LSTM remembers only the useful information, thanks to its cells having multiple *sigmoid* gates for information filtering.
The structure of an LSTM cell is presented below [20].

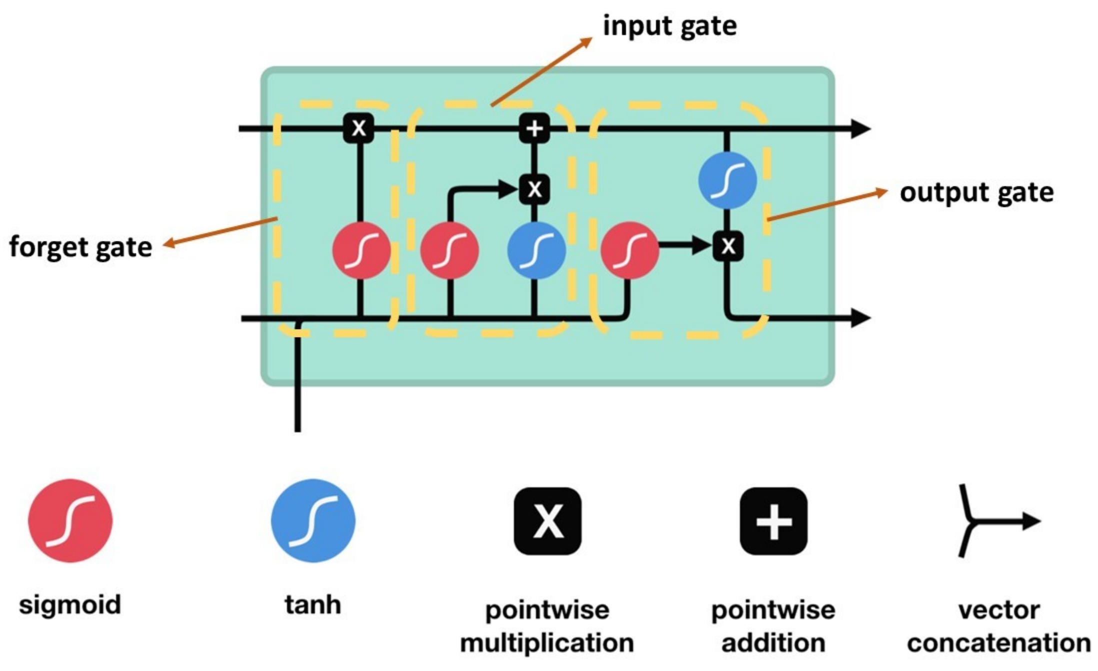


Figure 27: Structure of an LSTM cell.

The LSTM cells have gates, which helps the network remember past information.

- Forget gate: it uses a *sigmoid* function to filter information.
- Input gate: it uses a *tanh* and a *sigmoid* function (where the output of the former is the input of the latter) to update new information to the memory.
- Output gate: the current cell state passes through a *tanh* function while the input passes through a *sigmoid* function. The combined result will be the output which will get to the next cell.

Bibliography

[1] Edric John Cruz Nacpil, Zheng Wang, Rencheng Zheng, Tsutomu Kaizuka, and Kimihiko Nakano, "Design and Evaluation of a Surface Electromyography-Controlled Steering Assistance Interface", Sensors, 2019.
[2] Edric John Cruz Nacpil, Rencheng Zheng, Tsutomu Kaizuka, and Kimihiko Nakano, "A Surface Electromyography Controlled Steering Assistance Interface", Journal of Intelligent and Connected Vehicles, 2019.
[3] Edric John Cruz Nacpil and Kimihiko Nakano, "Surface Electromyography-Controlled Automobile Steering Assistance", Sensors, 2020.
[4] Zheng Wang, Satoshi Suga, Zhanhong Yan, Edric John Cruz Nacpil, and Kimihiko Nakano, "Adaptive Shared Steering Control System via Forearm Surface Electromyography Measurement", IEEE Sensors Journal, 2020.
[5] Petros Pandis, Joe Prinold, and Anthony Bull, "Shoulder Muscle Forces during Driving: Sudden Steering can Load the Rotator Cuff beyond its Repair Limits", Clin. Biotech, 2015.
[6] Masayoshi Wada and Fujio Kameda, "A Joystick Car Drive System with Seating in a Wheelchair", 18th IEEE International Symposium on Robot and Human Interactive Communication, 2009.
[7] Takahumi Minato, Yoshitoshi Murata, and Akimasa Suzuki, "Proposal of Automobile Driving Interface Using Strain Sensor for the Disabled People", 2015 Tohoku-Section Joint Convention of Institutes of Electrical and Information Engineers, 2015.
[8] Zheng Wang, Rencheng Zheng, Tsutomu Kaizuka, Keisuke Shimono, and Kimihiko Nakano, "Evaluation of Driver Steering Performance with Haptic Guidance under Passive Fatigued Situation", IEEE International Conference on Systems, Man, and Cybernetics, Budapest, Hungary, 2016.
[9] Aldo Perotto and Edward Delagi, "Anatomical Guide for the Electromyographer: The Limbs and Trunks", Charles C. Thomas Pub Ltd, 5th edition, 2011.
[10] Mohammed Z. Al-Faiz, Abduladhem A.Ali, and Abbas H.Miry, "A k-Nearest Neighbor Based Algorithm for Human Arm Movements Recognition Using EMG Signals", Iraq J. Electrical and Electronic Engineering, 2010.
[11] Zheng Wang, Zhanhong Yan, and Kimihiko Nakano, "Comfort-oriented Haptic Guidance Steering via Deep Reinforcement Learning for Individualized Lane Keeping Assist", IEEE International Conference on Systems, Man and Cybernetics, Bari, Italy, 2019.
[12] Steve Eddins, "Classify ECG Signals Using LSTM Networks", 6 August 2018, **URL**, Accessed 11 September 2020.
[13] Jason Brownlee, "How to Scale Data for Long Short-Term Memory Networks in Python", 7 July 2017, **URL**, Accessed 11 September 2020.
[14] Marcel Menner, Lukas Neuner, Lars Lünenburger, and Melanie Zeilinger, "Using Human Ratings for Feedback Control: A Supervised Learning Approach with Application to Rehabilitation Robotics", IEEE Transactions on Robotics, 2020.
[15] Norbert Wiener, "Extrapolation, Interpolation, and Smoothing of Stationary Time Series", New York: Wiley, 1949.
[16] Andreĭ Kolmogorov, "Stationary Sequences in Hilbert Space" (in Russian), Bull. Moscow Univ., 1941.
[17] Aapo Hyvärinen and Erkki Oja, "Independent Component Analysis: Algorithms and Applications", Neural Networks, 2000.
[18] Alaa Tharwat, "Independent Component Analysis: An Introduction, Applied Computing and Informatics", 2018.
[19] Chris Yoon, "Deep Deterministic Policy Gradients Explained", 20 March 2019, **URL**, Accessed 11 September 2020.
[20] Michael Phi, "Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation", 24 September 2018, **URL**, Accessed 11 September 2020.