ME2130 Research Project
Transport Engineering and Logistics
2017.TEL.8106

Strain gauge data analysis

Florian Hutten 4087585

Research Report

3mE - Transport Engineering & Logistics

July 15, 2017

Delft University of Technology

FACULTY OF MECHANICAL, MARITIME AND MATERIALS ENGINEERING
Department of Marine and Transport Technology

Mekelweg 2
2628 CD Delft
the Netherlands
Phone   +31 (0)15-2782889
Fax   +31 (0)15-2781397
www.mtt.tudelft.nl

Student:	F.M. Hutten		Assignment type:	Research/Computer
Supervisor:	Dr. X. Jiang		Report number:	2017.TEL.8106
Specialization:	TEL		Confidential:	No
Creditpoints (EC):	15		Theme:	2

Subject: **Development of data processing model for traffic specification of bridges.**

A lot of bridges in the Netherlands were built in the 1970-1980's and are suffering from fatigue and fracture issues. Therefore, it is essential to accurately predict the fatigue lifetime of those bridges in order to facilitate maintenance strategy planning and minimize the risk of catastrophic consequences. Although FEM models of bridge structures are well –established for application, the load cycles in FEM calculations is based on a generic assumed traffic pattern for all bridges, and thus do not represent actual traffic properly, so FEM model based calculation may not give an accurate prediction. To date, strain gauges have been widely deployed to measure the actual traffic loads on the bridge, however, a data processing model to retrieve useful information from abundant strain gauge data and transfer it into reliable traffic statistics is not yet well-established .

This research assignment aims to develop such a model to convert available strain gauge data of a bridge to a reliable output of traffic statistics. The development of such a model will make it possible to predict the fatigue life of bridge structures in a better and realistic way.

The scope of the research includes:

1) Set up the correlation between stain gauge data and available vehicle types.
2) Gain knowledge on MATLAB filtering and signal processing
3) Write MATLAB program which can identify correlations between strain gauge measurements and traffic specification
4) Validate the program using case studies.

This report should be arranged in such a way that all data is structurally presented in graphs, tables, and lists with belonging descriptions and explanations in text. The report should comply with the guidelines of the section. Details can be found on the website.

1 Summary

Fatigue crack propagation is a modern day problem for bridges which endure high traffic intensity. Especially on lanes which are designated for heavy vehicles. Cracks have been observed in these lanes years before such problems were expected to be happening. These observations have been made in bridges with orthotropic steel decks. One such bridge is the 'Van Brienennoordbrug' in The Netherlands, which has already undergone heavy maintenance due to unanticipated fatigue cracks. An array of strain gauge sensors were installed as to analyze the stress response to traffic. An initial analysis of these sensors was carried out by P. Kallitsas[3]. The results showed that analysis of certain parameters was possible, such as axle count, axle spacing and vehicle speed. Other parameters were however, highly uncertain, such as vehicle position, wheel count and vehicle weight calculation. These calculations were found to be time consuming and therefore inefficient to be carried out by hand. So in order to make large scale analysis possible, an algorithm would need to be designed which could analyze the data. This could enable the use of large strain gauge data sets to be used in determining the impact of traffic on the bridge.
The supplied strain gauge data has quite some noise, therefore a low-pass filter was used to filter out irrelevant data. The sensors were analyzed in two groups based on location on the bridge, due to the fact that different locations of the sensors provided different reactions to traffic input. The final step in tuning the data was normalization, so that different sensors could be properly compared to each other. Inspection of the strain gauge data pointed to the fact that peaks in the data, corresponded to vehicle axles crossing the sensor location. A peak detection algorithm with set boundary conditions was used to find peaks which belonged to axle crossings. To consolidate all the detected peaks, a handshake algorithm was used. It grouped peaks based on time-spacing between peaks, whereafter it decides the final point in time where the peak is located. The confirmed peaks are then again grouped based on which vehicle they belong to. This way an axle count of each vehicle can be established, as well as the point in time when it first crossed the sensors. With the point in time known for each vehicle, the speed of each vehicle is calculated by using the registered peaks from the sensors along the longitudinal direction of the road. With the individual speed of each vehicle known, the final calculation takes place to determine the axle spacing in meters. With all this available information the trucks are then classified according to the amount of axles and the spacing between them. A visualization of this gathered information can be seen in figure 1.1.

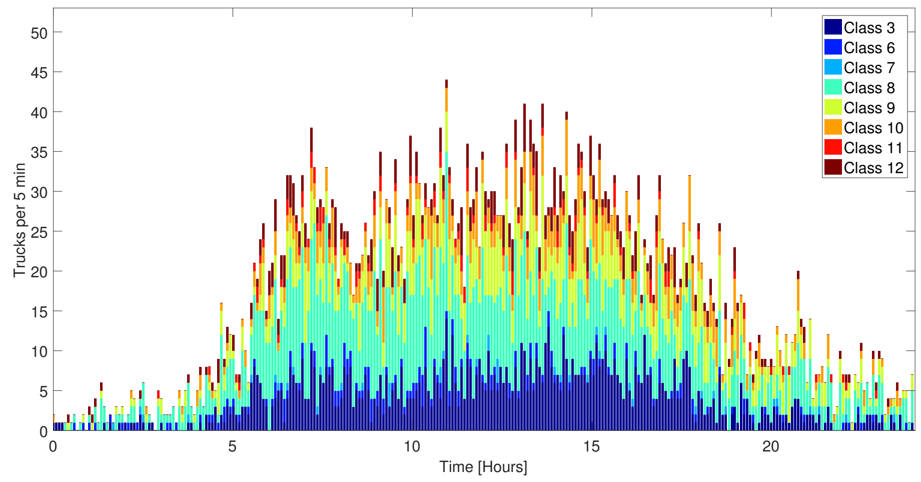


Figure 1.1: Visualization of general traffic statistics during 3 August 2015

In conclusion, the represented output by the algorithm shows promise in automatically analyzing large strain gauge data quantities. The data seems to represent the reality of the traffic class representation rather well. However this was not verified in the scope of this report. Therefore it is recommended to evaluate the reliability of the extracted data. Also analysis of further parameters is proposed, this could enable the use of the raw data to help understand the impact of traffic on fatigue life.

Table of Contents

1 Summary **iii**

2 Introduction **1**
2.1 Research question 1
2.2 Structure of report 2

3 General information **3**
3.1 Bridge construction 3
3.2 Sensor setup 4
3.3 Heavy vehicles 5

4 Data inspection **6**
4.1 Stiffener sensors 8
4.2 Deck sensors 9

5 Approach and strategy **10**
5.1 Peak detection 10
5.2 Vehicle identification 12
5.3 Speed and axle identification 14

6 Case Study **16**
6.1 Vehicle type distribution 16
6.2 Traffic intensity 18

7 Discussion **21**
7.1 Conclusion 21
7.2 Recommendations 22

Bibliography **a**

Appendix A Sensor layout **b**

Appendix B Peak detection **c**

Appendix C Speed estimation **d**

Appendix D Grouping peaks **e**

Appendix E Vehicle identification **f**

Appendix F Speed and axle spacing **g**

Appendix G Truck corrections **h**

Appendix H Raw data **i**

List of Tables

5.1 Vehicle speed with corresponding time-spacing 13
5.2 Example of trucks matrix 15

A.1 Detailed sensor coordinates on the bridge . . . . b

H.1 Raw data example from 3 August 2015 i

List of Figures

1.1 Visualization of general traffic statistics during 3 August 2015 iii

3.1 Orthotropic steel deck example [1] 3
3.2 Cross-section of the Van Brienennoordbrug 3
3.3 Crack locations on OSD [1] 4
3.4 Location of all the strain gauge sensors 4
3.5 Classification for vehicle types [7] 5

4.1 Overview of raw sensor data for sg4 6
4.2 Spectrogram for sg4 7
4.3 Comparison of filtered vs unfiltered data from sg4 7
4.4 All strain gauges at the same longitudinal line 8
4.5 Filtered data for sg1, sg2, sg3, sg4, sg5 and sg6 8
4.6 Filtered and normalized data for sg1, sg2, sg3, sg4, sg5 and sg6 9
4.7 Filtered and normalized data for sg7, sg8, sg9, sg10, sg25 and sg31 9

5.1 MATLAB flowchart 10
5.2 sg4 readings over 100 seconds 10
5.3 Peak prominence [6] 11
5.4 Finding peaks in the dataset 11
5.5 Confirmed peaks plotted against sensor data 12
5.6 Visualization of time-spacing between 2 vehicles 13
5.7 Visualization of time-spacing between 2 vehicles 14

6.1 Axle count for 3 August 2015 16
6.2 Axle count for 4 January 2016 17
6.3 Axle count for 3 August 2015 17
6.4 Axle count for 4 January 2016 18
6.5 Visualization of peak detection error 18
6.6 Visualization general traffic statistics during 3 August 2015 19
6.7 Visualization general traffic statistics during 4 January 2016 19
6.8 Vehicle types per 5 minutes on
3 August 2015 20
6.9 Vehicle types per 5 minutes on
4 January 2016 20

2 Introduction

Reliably predicting infrastructure design lifetime is a hard and time intensive process. Such predictions are made with complex FEM models with a high amount of detail. These models require a large amount of information which is not always fully available, such as load cases and Material-Material interactions across time and with different temperatures. This can result in inaccurate predictions about the structural integrity over time and thus influence the safety and durability of the structure.

To guarantee the safety of the structure, structural health monitoring is a crucial step to identify weaknesses and plan for necessary maintenance intervention. The key to maintaining these large and numerous infrastructure projects would be a process that is reliable enough to provide the necessary data as well as be smart enough to handle multiple types of these structures. This together should be able to conclude what the best maintenance strategy would be to increase safety as well as lifetime of the infrastructure.

The reliability of infrastructure is important for several types of infrastructure, such as railroads, bridges and tunnels. This project will be purely focused on bridges, to be more precise, the 'Van Brienenoordbrug' in The Netherlands. This is a choice made based on available data from TNO as well as a previous Thesis written by P. Kallitsas [3].The data made available by TNO consists of strain gauge measurements over the course of roughly two years. They were installed as a demo project for Rijkswaterstaat to investigate and improve decision making based on non-destructive testing techniques. The Thesis by Kallitsas in an in depth look into what type of information can be extracted from raw strain gauge data. He states that when analyzing strain gauge data it is possible to extract information such as speed, location and possibly the weight of large vehicles.

2.1 Research question

Valuable information can be extracted from the raw strain gauge data, such as axle count of vehicles, speed of vehicles and the location of vehicles on the road. However this information is extracted using manual calculations of several time intervals. This would be inefficient to apply to the multiple years of data that is available. As such an approach needs to be found which enables the processing of large sets of strain gauge data. This would then enable insight into the possibilities that lie within raw data.

How can available bridge strain gauge data be automatically analyzed to identify heavy vehicle traffic statistics.

To further the clarification of the main question, several sub-questions are proposed as a baseline to approach answering the main question.

- what steps need to be taken to be able to automatically compare all different available strain gauges?
- How can individual vehicles be identified from the dataset?
- What parameters can be reliably extracted from the dataset?

This analysis and automation will be done in the MATLAB, which is a numerical computing environment. This is used with the implementation of the signal processing toolbox, as to better analyze the large quantities of data.

2.2 Structure of report

The report will firstly discuss the background information, followed by the immediate observations which can be made from the available data. It will then continue to discuss the approach on how to analyze the data. This is done by describing fragmented parts of the algorithm used to find the relevant statistics. These statistics will then be visualized and discussed in the next chapter in the form of a case study. The report will then finish with its final chapter, which holds the conclusion, discussion and further recommendations.

3 General information

This chapter is dedicated to help interpret the available data. This will include the effects of different types of traffic on bridges as well as information on how the bridge itself is constructed.

3.1 Bridge construction

To understand the different types of impact that traffic has on the bridge, it is important to understand what type of bridge is being analyzed. The Van Brienennoordbrug is a bridge with an Orthotropic steel deck (OSD), an example of such a structure can be found in figure 3.1.

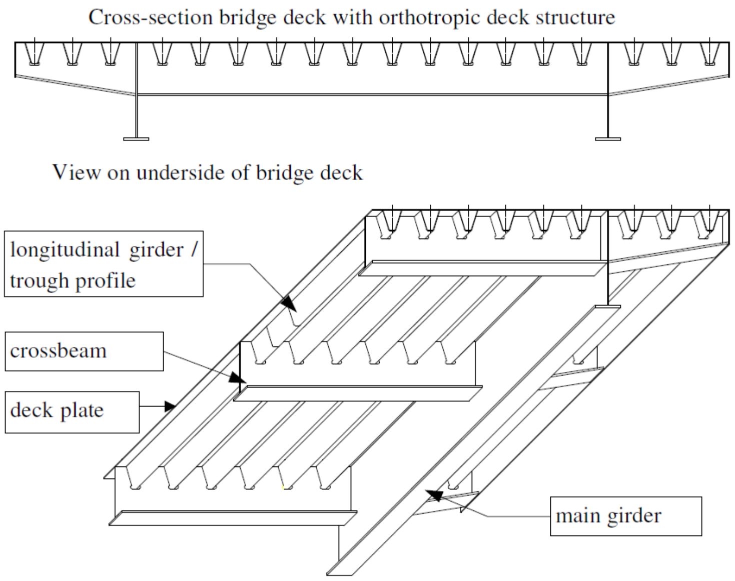

Cross-section bridge deck with orthotropic deck structure
View on underside of bridge deck
longitudinal girder / trough profile
crossbeam
deck plate
main girder

Figure 3.1: Orthotropic steel deck example [1]

The Van Brienennoordbrug is a bridge located in Rotterdam across the Nieuwe Maas. The bridge hosts room for 12 different lanes of traffic for the A16 highway. The bridge has a total length of 1320 meters, with the longest single span being 300 m. The bridge has a separate drawbridge part, which spans around 60 m. Figure 3.2 shows the cross-section of the main deck of the bridge. The traffic intensity across the bridge is one of the highest in the Netherlands. When the bridge started to show signs of fatigue damage in the drawbridge, the whole drawbridge needed to be replaced. This replacement was a highly cost intensive action, but had to be done because appropriate targeted fatigue renovation techniques were not known.

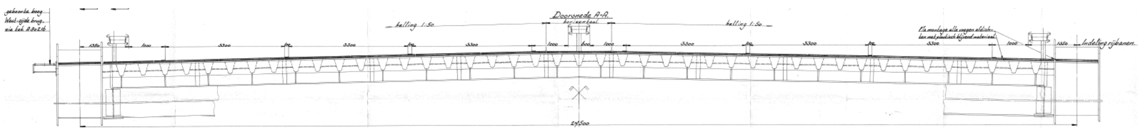


Figure 3.2: Cross-section of the Van Brienennoordbrug

The reason for researching the effect of traffic on the structural integrity of the bridge, is because multiple bridges have shown signs of fatigue damage before such damage was anticipated. With the highest rate of occurrence being in the heavy vehicle lanes. The fact that these cracks started to occur in the Van Brienennoordbrug after only 7 years of service is the most perplexing [5]. Thus the interest arose to specifically collect data on heavy vehicle lanes to help understand the cause of these fatigue cracks. Figure 3.3 shows the locations on the deck where fatigue cracks occur. They mostly occurred at the intersection with the crossbeam, which coincidentally is also the most dangerous location regarding safety of the structure.

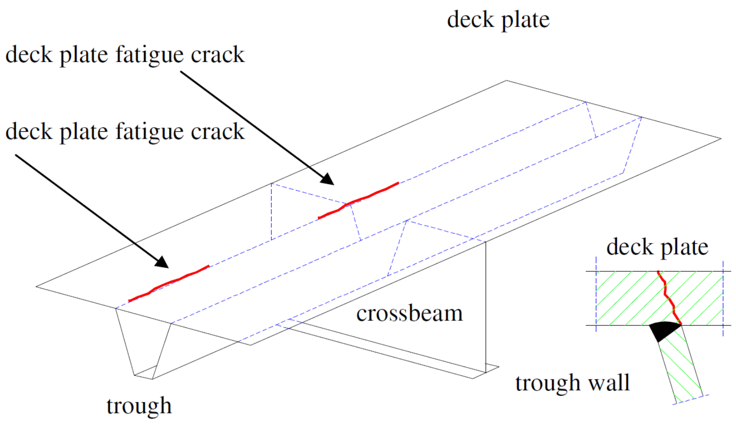


Figure 3.3: Crack locations on OSD [1]

3.2 Sensor setup

Based on all the information available a setup was chosen for the sensors that were to be implemented in the bridge. They would have to be located in such a way that as much valuable data as possible could be extracted. Figure 3.4 shows the complete layout of all implemented strain gauges. The figure shows one lane in the middle of the bridge, which is the lane designated for heavy vehicle traffic. The location of the sensors are centered around the area where the wheels of the vehicles have contact with the asphalt, as to have the best readouts. Some sensors are located directly underneath the deck of the road, while others are located at the bottomside of the stiffeners below the deck. The detailed coordinates of all the sensors can be found in Appendix A.

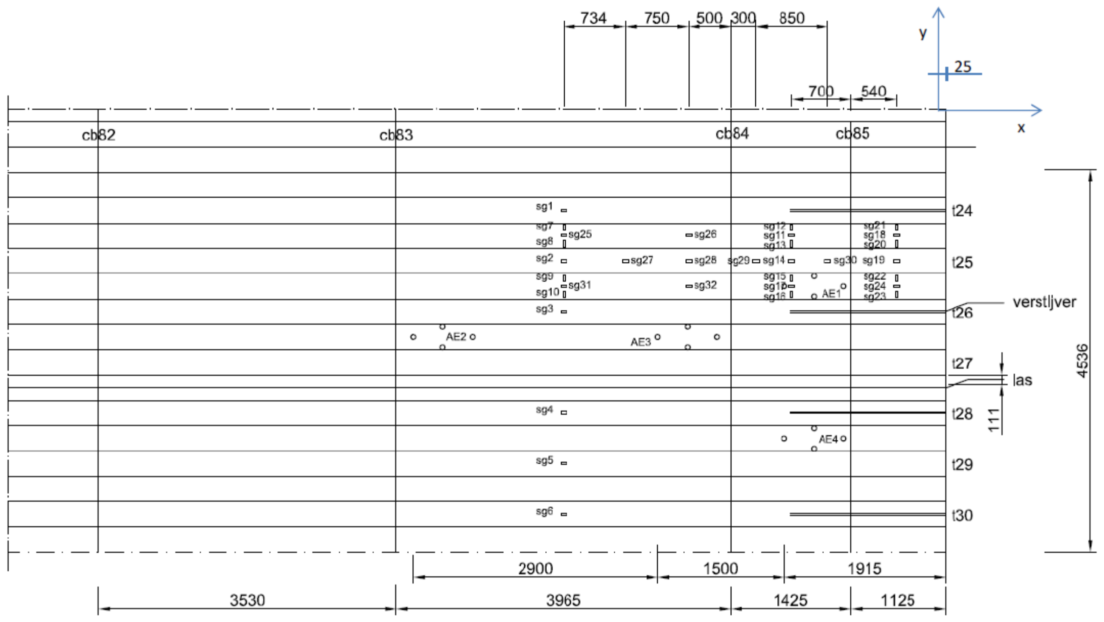
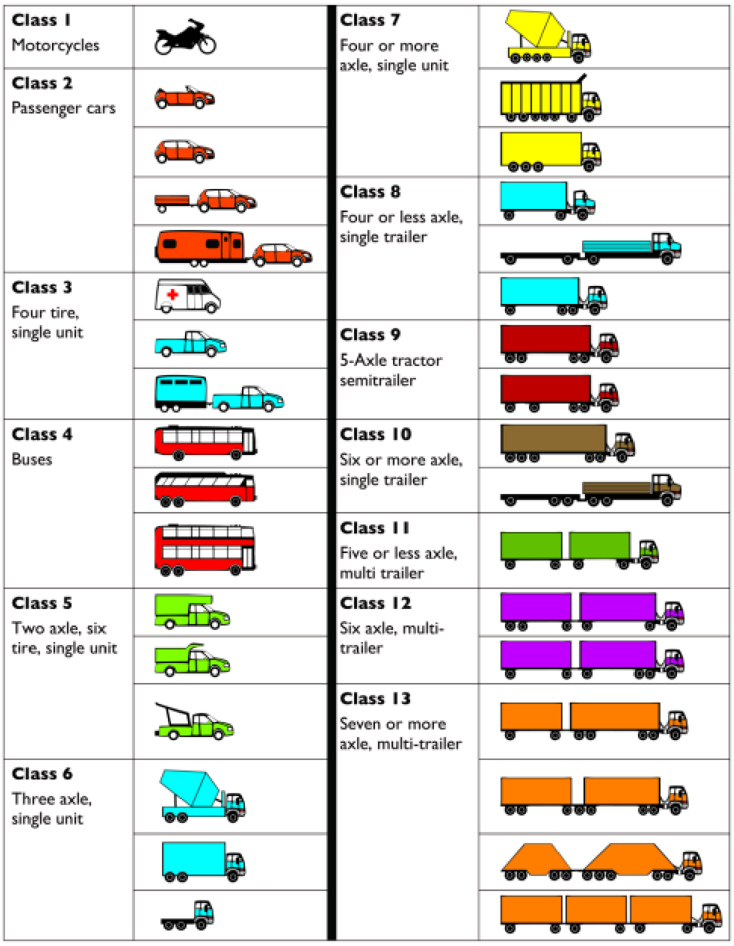


Figure 3.4: Location of all the strain gauge sensors

3.3 Heavy vehicles

For this research only heavy vehicles will be examined based on the available sensor data. This was already pre-established because of the high amount of structural damage that was found on especially the heavy vehicle lanes. The vehicles traversing this heavy vehicle lane can be put into several classification as to identity different vehicle types. A classification of such a kind can be found in figure 3.5. In this research class 1, 2 and 3 shall not be looked into to, as mentioned before. Also vehicle class 4 will not be differentiated from class 6, due to the fact that both vehicles types have the same footprint. The main vehicle types that can be found are semi-trailers, buses and construction vehicles. The difference between most listed trucks is based on whether they are one rigid construction or have a separate semi-trailer connected to a tractor unit as well as the total amount of axles that support the vehicle and trailer.

The complete vehicle is logically carried by its axles. The wheels on the axles are logically the only contact point between the load of the vehicle and the deck of the bridge. Therefore they will determine the load distribution of the truck on the bridge. As can be seen in figure 3.5, there is a high variance in axle configurations. It is therefore important to determine which type of vehicles have a high occurrence, as to create a reliable traffic profile.



Class 1 Motorcycles	**Class 7** Four or more axle, single unit
Class 2 Passenger cars	**Class 8** Four or less axle, single trailer
Class 3 Four tire, single unit	**Class 9** 5-Axle tractor semitrailer
Class 4 Buses	**Class 10** Six or more axle, single trailer
Class 5 Two axle, six tire, single unit	**Class 11** Five or less axle, multi trailer
Class 6 Three axle, single unit	**Class 12** Six axle, multi-trailer
	Class 13 Seven or more axle, multi-trailer

Figure 3.5: Classification for vehicle types [7]

4 Data inspection

Now that the scene has been set and the objective is known. It is important to learn and identify how the different strain gauges react to traffic on the bridge. The strain gauges installed by TNO have a sampling frequency of 1200 Hz and record data for all 24 hours during a day. The data sets are split up into fragments of 5 minutes, as to make file size manageable. All sensor and temperature data in these 5 minutes is written in single data files. Figure 4.1 shows an example plot of a single unprocessed data set of a strain gauge over the course of 30 seconds. In this case it is strain gauge 4, based on the overview in figure 3.4. From now on strain gauges will be denoted as 'sg#'. The plot and all following plots are made using MATLAB.

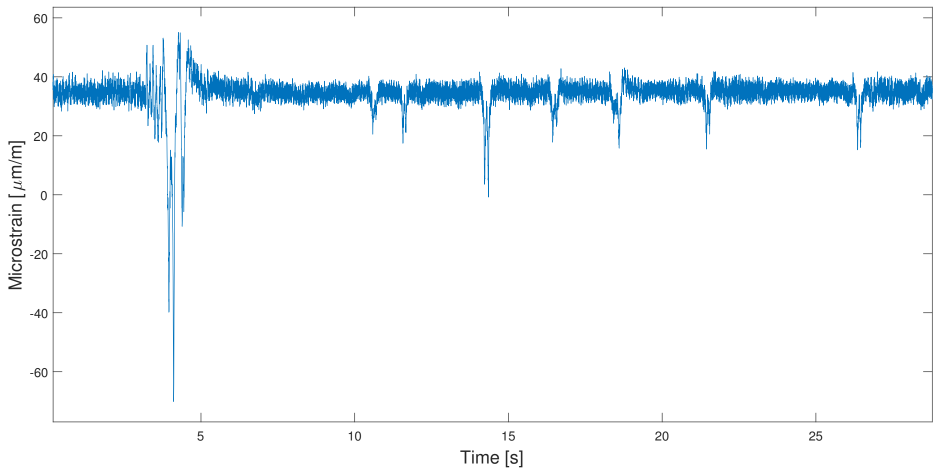


Figure 4.1: Overview of raw sensor data for sg4

From the very first observation it can be seen that there is quite some noise in the data set. The noise can be identified in figure 4.1 by looking at the data between the peaks. In an ideal scenario one would expect this to be a straight line and not a waveform. To reliably analyze the data it will first need to be filtered. To analyze the noise frequencies in the signal, spectrogram plots were made for all the sensors. This way an optimal filter strategy can be formed. Such a spectrogram plot can be found in figure 4.2, this is the spectrogram plot over 2.5 minutes from sg4.

From this figure it can be deduced that all valuable data is present below the 100 Hz. An ideal filter to remove the noise from the signal would be a low-pass filter. The cutoff frequency is then established at 100 Hz to remove the amplified noise which can be seen in the spectrogram. In figure 4.3 the result is shown of the filtered data compared to the original raw data as seen in figure 4.1. An immediate difference can be seen in the reduced noise between the peaks. This process is then repeated for all different sensors in the bridge.

Now that the data is filtered, it opens up the possibility to compare the data of different sensors. To be able to do that, a baseline is needed on how certain sensors respond to input. It is expected that not all sensors give the same type of readouts due to the fact that they are located on different parts of the bridge structure (below the deck or below the stiffeners). Also, not all sensors are located at the same longitudinal position on the bridge. Meaning that between some readouts a time-shift is expected because of the distance between the sensors. Figure 4.4 shows a data plot of all the sensors at the same longitudinal location on the bridge. Two major observations can be made from this plot. First of all it can be seen that the data needs to be normalized, so that all readouts will have a zero mean. Secondly there is a clear distinction between positive and negative readouts. A further analysis reveals that sensors located at the bottom of the stiffeners will give negative peaks, while strain gauges located at the deck will have positive peaks. This can be explained by the fact that one type of sensor is under compression forces, while the other

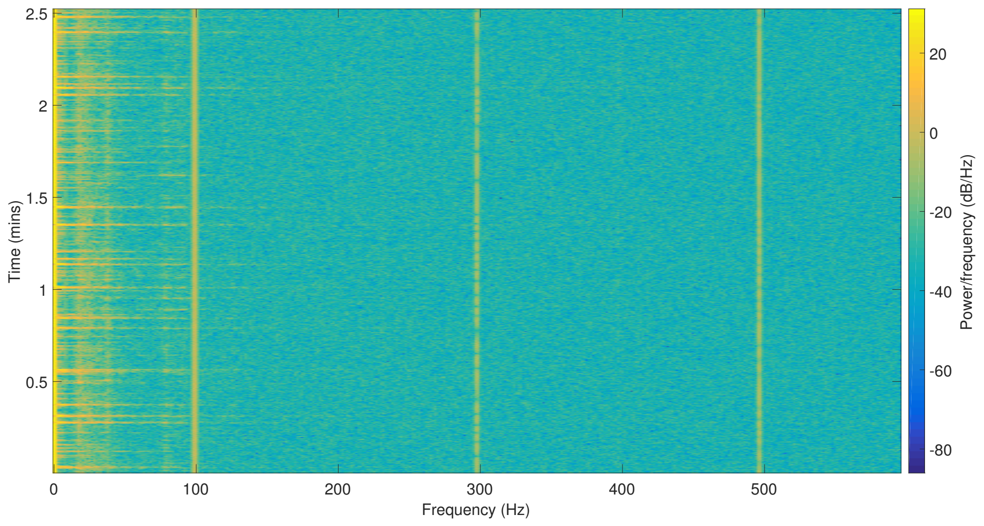


Figure 4.2: Spectrogram for sg4

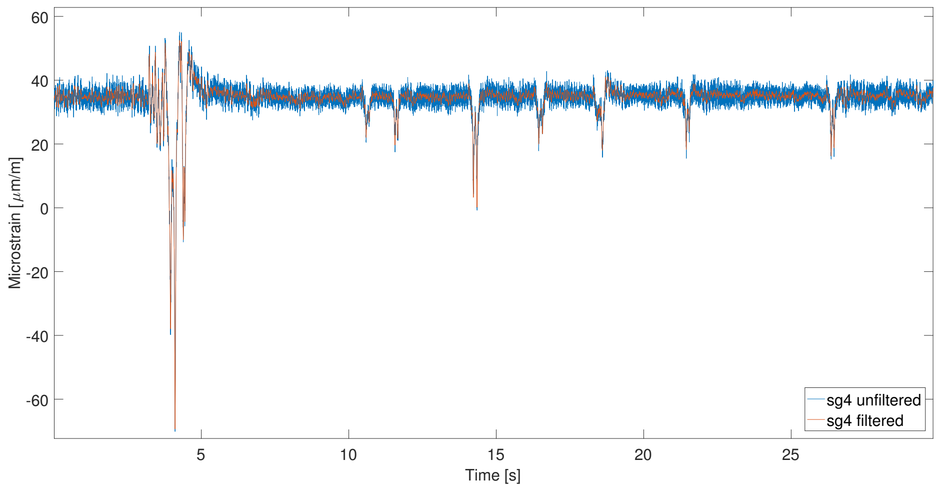


Figure 4.3: Comparison of filtered vs unfiltered data from sg4

is under tension. Now that it has been established that there is a separation between the type of readouts that the sensors give, the decision is made to process the sensors separately as well.

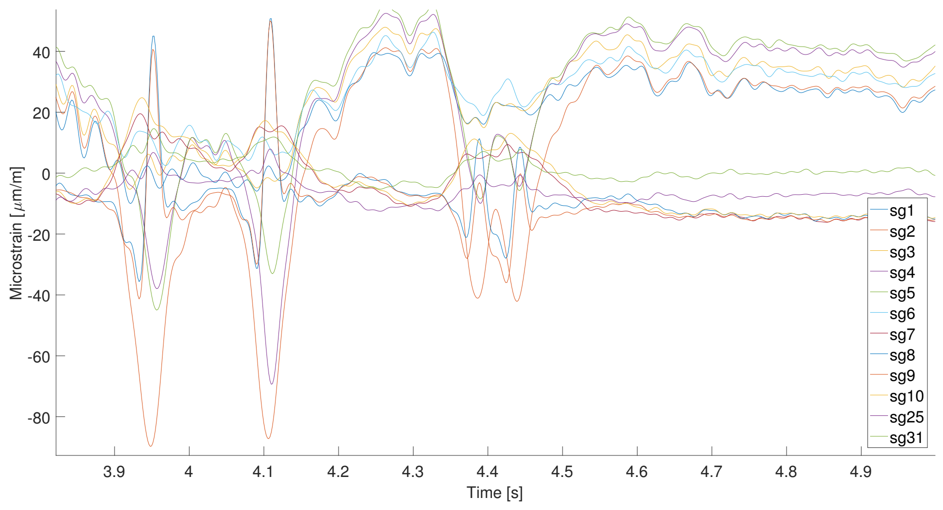


Figure 4.4: All strain gauges at the same longitudinal line

4.1 Stiffener sensors

For this comparison we will be looking into sensors: sg1, sg2, sg3, sg4, sg5 and sg6. The reason these sensors are in this selection is because they are all located at the bottom of the stiffeners of steel deck, as well as on the same longitudinal location. The plot of all the sensor readouts can be found in figure 4.5.

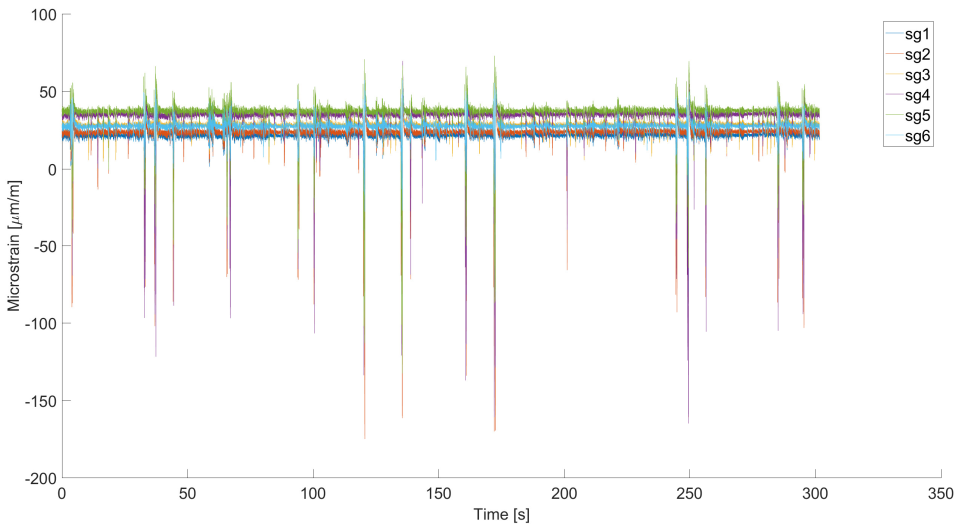


Figure 4.5: Filtered data for sg1, sg2, sg3, sg4, sg5 and sg6

As mentioned before, the high peaks show overlap between the different sensors. Though it can be observed that the sensor data is not normalized with a zero mean. So in order to fairly compare the different sensors, the data was normalized using formula 4.1. Where $\overline{x}$ is the mean of the signal over 5 minutes. Figure 4.6 shows the the data of the same sensors, but then with normalized values. With the normalization and filtering done, it is now possible to compare the data and try and analyse what data is relevant and what is not.

$$x = x - \overline{x} \tag{4.1}$$

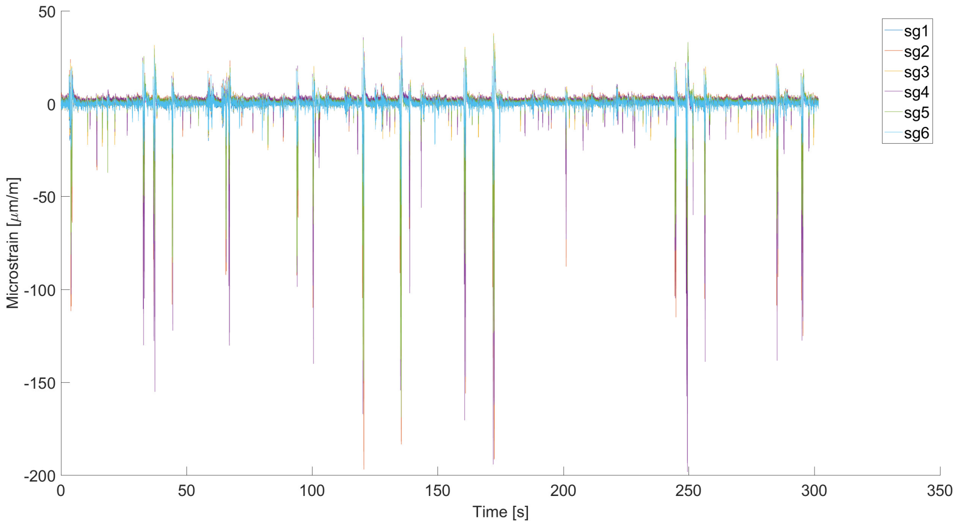


Figure 4.6: Filtered and normalized data for sg1, sg2, sg3, sg4, sg5 and sg6

4.2 Deck sensors

The sensors located at the deck of the bridge are: sg7, sg8, sg9, sg10, sg25 and sg31, as can be seen in figure 4.4. The data for these sensors also needs to be normalized. This is again done using formula 4.1. The results of all these filtered sensors can be seen in 4.7. Comparing this figure to figure 4.6, confirms that it was a good idea to separate the two types of sensors locations. It can be clearly seen that these sensors have a very high positive readout, as well as a noticeable negative readout. This should be taken into account with the creation of the algorithm, as to avoid misinterpreting data due to the generalization of all sensors.

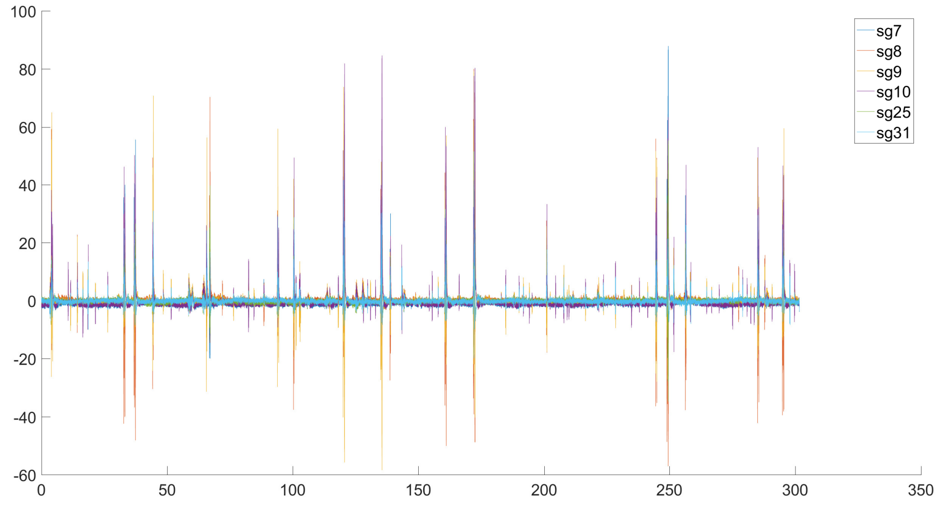


Figure 4.7: Filtered and normalized data for sg7, sg8, sg9, sg10, sg25 and sg31

5 Approach and strategy

With the filtered and normalized data, an approach can be formulated. It is important to determine what type of relevant information can be extracted from the raw data. At a first glance it can be observed that the peaks represent traffic crossing the sensors. This chapter will discuss the chosen relevant data to be extracted, as well as the techniques used to achieve them.

As this research is dedicated to set up a baseline which can be used for further analysis, the choice was made to try and extract the most fundamental information. This includes the identification of vehicles, discerning what type of vehicles they are, the speed of the vehicles as well as their axle count with the spacing in between the axles. Figure 5.2 shows a segment of sensor data, the figure shows a distinction between light and heavy vehicles. This is important because this research focuses on heavy vehicles as mentioned in Chapter 2. Which makes these readouts very suitable for this analysis.

Before any analysis can be carried out by the program, it first needs to be initialized, thereafter the analysis of the files is possible. Figure 5.1 shows a schematic overview of the structure of the MATLAB program.

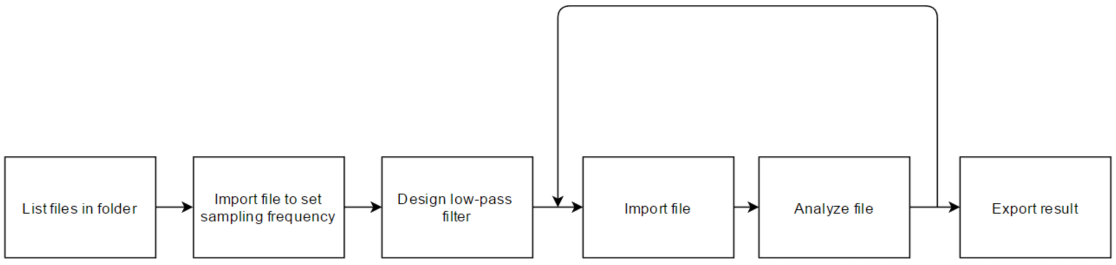


Figure 5.1: MATLAB flowchart

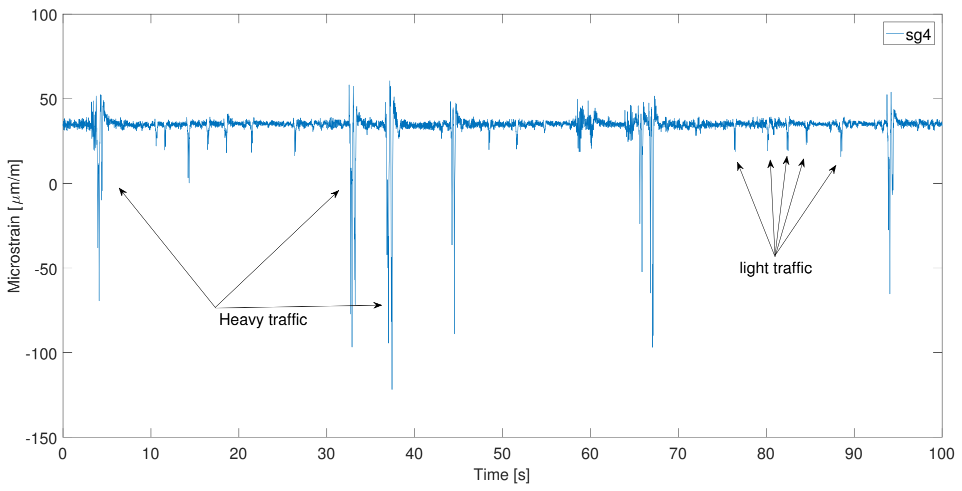


Figure 5.2: sg4 readings over 100 seconds

5.1 Peak detection

The first step in extracting relevant information from the data is identifying the individual peaks from the heavy vehicles. These individual peaks represent the load peaks from the crossing of axles of the vehicles. However what can already be seen in figure 4.4, is that the sensors are highly sensitive to location. Meaning it is not guaranteed that a sensor will read a peak when a vehicle crosses its path. It is therefore imperative to collect peak data from as much sensors as possible to form a reliable conclusion. The code used to execute this task is found in Appendix B.

As can be seen in the code, certain boundary conditions are applied to find the peaks. The first condition is the minimum peak height, this represents the minimum microstrain required to be accepted as a peak. This term is introduced to exclude all the small vehicle traffic, such as cars and small vans. These type of vehicles are not to be included in this research. The second condition is the minimum peak prominence, the peaks prominence is best displayed using figure 5.3.

The figure shows the peak prominence of each peak using colored areas. The prominence is determined by horizontal lines from the peak top until it intersects with another line or the end of the graph. Then find the minimum on each side of the peak, then the prominence is defined as the height from this minimum to the peak height. This condition is used to identify how unique a peak is given its surrounding peaks. This prevents the program from identifying multiple peaks within a short span due to vibration. However the constraint cannot be set too tight, as that would cause problems with triple axle setups. Such a triple axle setup will have three rather similar peaks which should all be identified. The third constraint used in the 'findpeaks' formula is the minimum peak distance. This constraints ensures that two following peaks should be at least some seconds apart, again this is to prevent the identification of multiple peaks where only one is relevant. The output of the algorithm in Appendix B contains several data points, these are the height, location and prominence of the found peaks. These data points are then stored in cell structures with the variable name corresponding to the sensor number. Figure 5.4 shows a visualization of the found peaks of the six sensors on the stiffeners. The found peaks are visualized by the stars in the graph. As can be seen only three sensors show relevant data in this case.

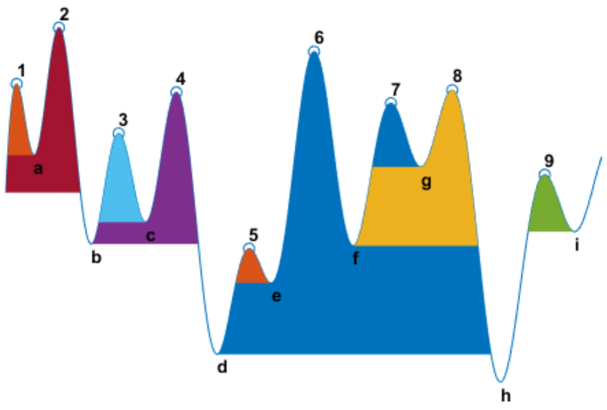


Figure 5.3: Peak prominence [6]

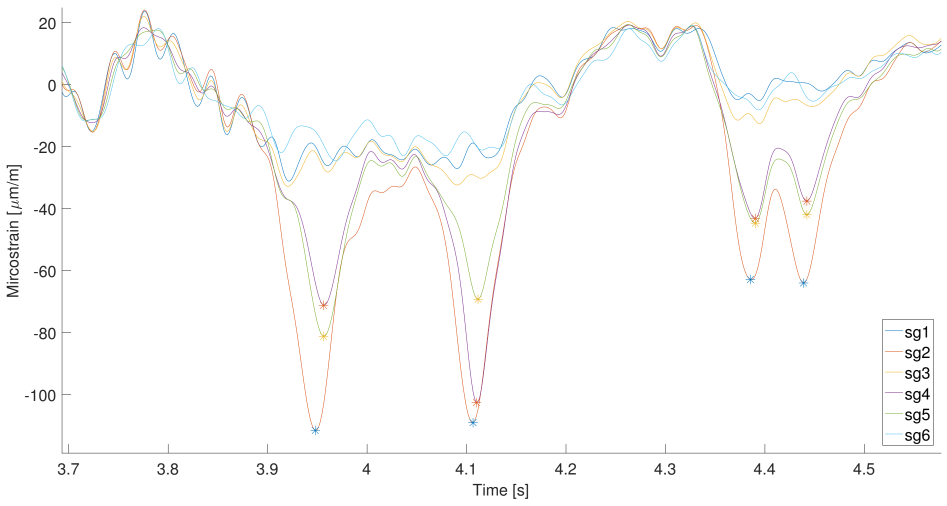


Figure 5.4: Finding peaks in the dataset

Several peaks have now been found, however this is not the type of result that is directly useful. The information that is wanted is just the time point of the axles crossing the sensors. Though as can be seen in figure 5.4, not all maxima are located at the same point in time. This could be the result of how the deck reacts to loads. The elastic deformation of material at the location of the sensors is not always equally timed due to sensor location or because of misaligned sensors on the bridge. So it is expected that the first peak in a series is closest to the load location and point in time, whereas lagging peaks are expected to belong to sensors that were further away from the peak load.

Therefore a consensus has to take place in order to determine the timing of the peak. This is also a useful tool to eliminate false-positives, by requiring that several different sensors detect a peak at a certain point in time. The grouping of found peaks can be achieved by creating small search intervals with a width of ΔT. This search interval needs to be dynamic, as slow moving traffic could result in a larger variance between the locations of the peaks. Therefore an estimate needs to be made on the speed of the traffic as to determine these search intervals. Appendix C shows the code used to determine the average vehicle speed over 5 minutes.

With the vehicle speed known, the search interval can be used to group the peaks from different sensors. For this search interval it is estimated that the minimum axle spacing is at least 1 meter [2]. With the use of formula 5.1 the interval is calculated.

$$\Delta T = \frac{1}{speed} \tag{5.1}$$

Using this dynamically adjusting interval, it is possible to group the detected peaks from different sensors together. This is done using the code Appendix D. It sorts all the peak data from the different sensors in one matrix. It then follows a loop which detects how many peaks fall into the established ΔT interval. It then takes the most prominent peak from the set and saves that as a confirmed peak. If only one sensor detects a peak within the interval, the peak is ignored. This action was taken after inspection, because it was seen that all peaks with only one sensor confirmation were not relevant and were assumed to be measuring errors.

After the execution of this code, one matrix remains with confirmed peaks. This matrix contains the location, height and the amount of peaks grouped together. Figure 5.5 shows the final result of code to find the peaks, the data of the six sensors is plotted against the confirmed peaks.

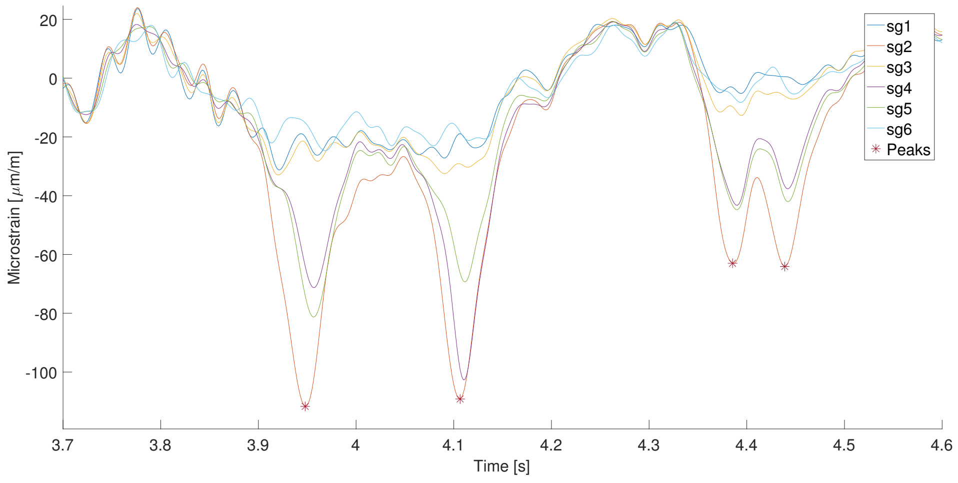


Figure 5.5: Confirmed peaks plotted against sensor data

5.2 Vehicle identification

The next step in the analysis is to identify the individual vehicles in the data. With the aforementioned peak detection, individual peaks are known, however it is not known which peaks belong to which vehicle. Therefore an algorithm needs to be made that groups certain peaks together which are deemed to belong to one vehicle. The best way to identify where a vehicle ends and where another starts would be to analyze the time difference between the peaks. There is a road law in The Netherlands which states that vehicles should always have 2 seconds of time-spacing between vehicles. However this would be a very optimistic minimum time-spacing, because not everybody abides by this law. Therefore a lower time-spacing would need to be determined. Such a minimum can be made using the maximum axle spacing used in vehicles, with the knowledge of the speed of the vehicle the time-spacing could be determined. The maximum axle spacing allowed is assumed as 12 meters based on data from the RDW [8]. Then the time-spacing for 12 meters with different speeds is calculated and a trade-off is made with the law citing 2 seconds. The time-spacings are not equal to the 12 meter timing to prevent measuring errors or speed variations from impacting the results. A summarized reference of values can be found in table 5.1. The table is used as a baseline, which is then interpolated in the code. This is done as to create a higher density of data points, to increase the accuracy of chosen time-spacing. It was chosen that higher speeds would allow for smaller time-spacing between vehicles. This is due to the fact that 12 meters is covered faster at higher speeds. Therefore the accuracy can be increased by allowing for smaller time-spacings between vehicles. The reverse is true for vehicles at lower speeds.

Table 5.1: Vehicle speed with corresponding time-spacing

Speed [km/h]	0	20	40	60	80	100	120	140	160
Time-spacing [s]	1.6	1.5	1.4	1.3	1.2	1.1	1	0.9	0.8

Figure 5.6 visualizes what the program needs to find. It should detect which axle is the last axle of a vehicle.

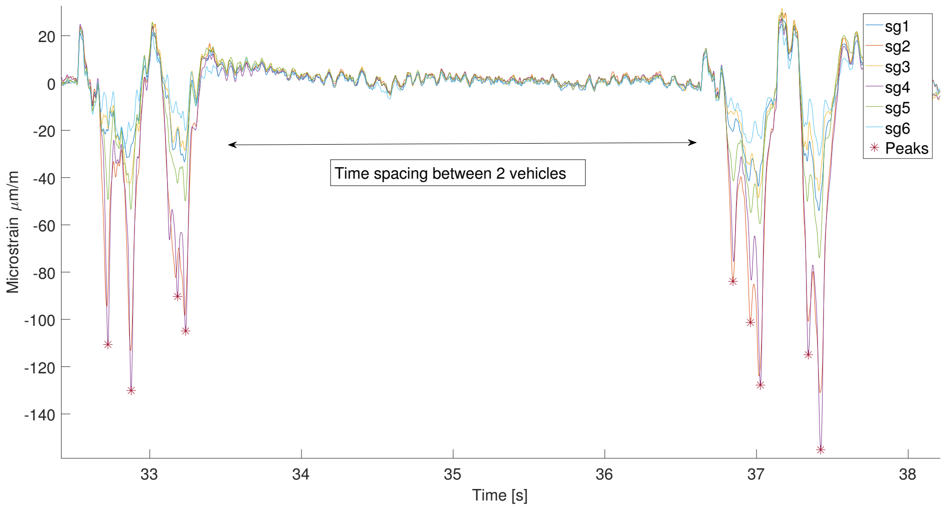


Figure 5.6: Visualization of time-spacing between 2 vehicles

To identify the vehicles, the code from algorithm E is used. It calculates the difference time-spacings between peaks and interprets whether the peaks belong to the same vehicle or not. It then saves the data in a cell structure with the name 'trucks'. The matrices within the cell contain the axle count per vehicle as well as the time the first axle crossed the sensor. This is done as to identify 'where' the vehicle is located within the data. This can then be used to more accurately determine the speed per vehicle instead of the speed over the whole of 5 minutes. This can also be used to log what type of vehicle is traveling at what time during the day or even season. It is therefore important to log when a vehicles passes the sensors. This cell will later be used to store additional information.

5.3 Speed and axle identification

With the added knowledge of when a vehicle passes the sensor, it is now possible to determine the speed of every single vehicle passing the sensors. In order to get accurate results for the speed of each vehicles, multiple sensors need to be used. In figure 3.4 all the available sensors can be seen. After inspecting all sensors, it was determined that not all sensors contained enough relevant data. A lot of sensors were not able to detect peaks which were detected by other sensors. Therefore the choice was made to use a select group of sensors which showed high responsiveness to loads of vehicles. These sensors were sg27, sg28 and sg29.

Figure 5.7 shows a plot of a limited time-slot for sg2 as well as the above mentioned sensors. It can be clearly seen that there is a time-shift between the different sensors. In order to calculate the speed, the time-shift between the sensors needs to be determined. With the knowledge of the different distances between the sensors it would then be possible to determine the speed. With the previous code all peaks are already confirmed for all the sensors which share the lateral location of sg2. With the additional knowledge where the first peak for each vehicle resides, it is possible to scan the other sensors for peaks which happen before the first peak of each vehicle. With the knowledge that we know a peak should reside within a certain time frame, a relaxation of the formula to find peaks is used for sg27, sg28 and sg29. Which allows for a smaller minimum peak height as well as a lower minimum peak prominence. This ensures that there is a lower chance that a peak is missed while it should have been found.

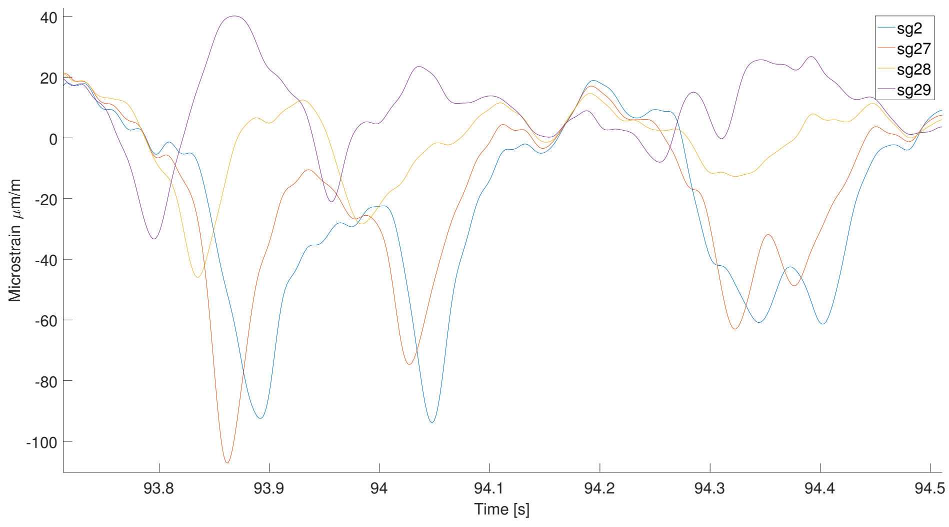


Figure 5.7: Visualization of time-spacing between 2 vehicles

The code for calculating the speed per vehicle can be found in Appendix F. The first action is to subtract the vector with locations of the peaks from the location of the first peak of a truck in the form of a scalar. The results is a vector containing the time-spacing between a truck and all the peaks from sensors 27, 28 or 29. It then checks whether the vector contains positive values, if so it takes the smallest positive value as the time-spacing between the two relevant peaks. Then the distance between the sensors is divided by the time-spacing to calculate the speed. This is done for all three sensors. It could happen that a necessary peak is not found, which results in a very low speed due to the fact that then a peak would be detected which is farther away. When this occurs it is filtered out by checking if the value is smaller then half the highest speed. The highest speed is used as a reference point because that has the highest probability of being the correct speed. This is the case because higher speeds are the cause of a close proximity of peaks, which means two peaks are probably a pair. While low speeds indicate a higher time-spacing which can be the result of a closer peak not being detected. The remaining speed results are then averaged and stored in the matrix 'trucks', which contains all relevant information about each truck.

The second part of the code in Appendix F, contains the code to calculate the axle spacing of each truck. It looks up the first peak of each truck in the vector which contains the validated peaks. Then with the use of the axle count, it finds the other peaks belonging to the vehicle. The time-spacing between each peak is then multiplied by the speed to find the axle-spacing in meters. The axle spacing is then stored in the 'trucks' matrix, an example of this matrix can be found in table 5.2.

Table 5.2: Example of trucks matrix

Axle count [-]	First peak location [s]	Speed [m/s]	Axle spacing [m]			
			Axle 1-2	Axle 2-3	Axle 3-4	Axle 4-5
3	4.10	27.46	7.65	1.46	0	0
2	14.21	36.62	4.40	0	0	0
4	32.72	23.26	3.57	7.10	1.24	0
5	36.84	23.28	2.68	1.51	7.35	1.90
2	44.24	25.11	6.79	0	0	0

After analysis of the output data provided by the combination of the above mentioned code. It can be found that sometimes errors are made regarding the grouping of multiple vehicles. This can happen because of the lenient time-gap that is accepted as the spacing between vehicles. However, now that a good estimate can be made about the inter-axle spacing, it is possible to separate vehicles due to inter-axle spacing which is too high. Appendix G shows the code used to achieve this. It first finds all axle spacing's in the trucks matrix which are higher then 12 meters, then the location of this axle in the matrix is stored. It detects whether a whole new truck drives after this gap, or whether it is a single standalone peak. When a standalone peak is detected, the unrealistic data is removed. If more peaks are present after the large gap, a new row is formed to state the information about the new trucks. All necessary data is found (speed,location,axle count) and stored in this new row. An exception is made to check whether multiple exceeding values are found in a single row.

6 Case Study

This chapter will show a visualization of the data extracted using the algorithm to analyze the sensor data. The results will be based on all the data from 3 August 2015 and 4 January 2016. An explanation will be given about the data using visualizations in the form of bar charts and graphs. An example of the used raw data can be found in Appendix H. The appendix shows the first 0.014 seconds of the data for the first few sensors on 3 August 2015.

6.1 Vehicle type distribution

Different types of vehicles cross the bridge during the day. This can first of all be defined by counting the axles. This gives an initial overview of which type of vehicles cross the bridge. Figure 6.1 and figure 6.2 show an overview of the axle count of the vehicles during a single day. Of course 2 axle vehicles are not over-represented due to the fact that smaller vehicles are not registered, unless they have a significant load on board.

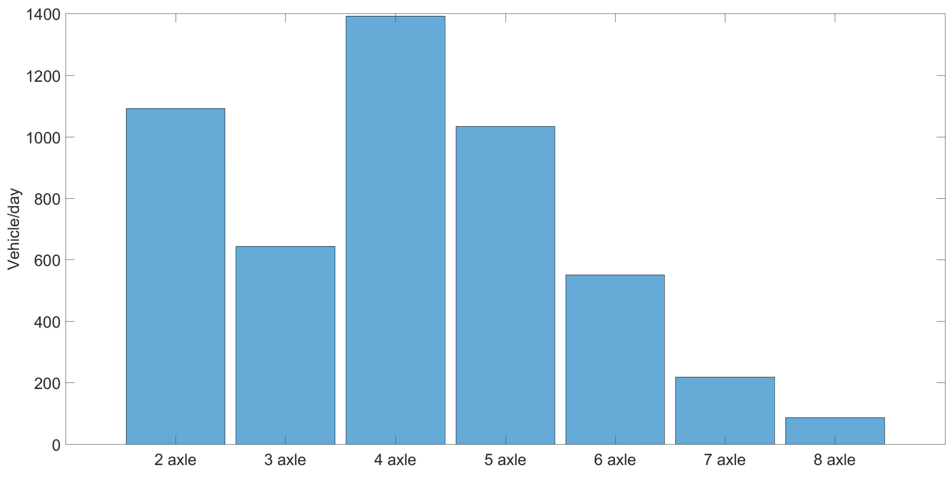


Figure 6.1: Axle count for 3 August 2015

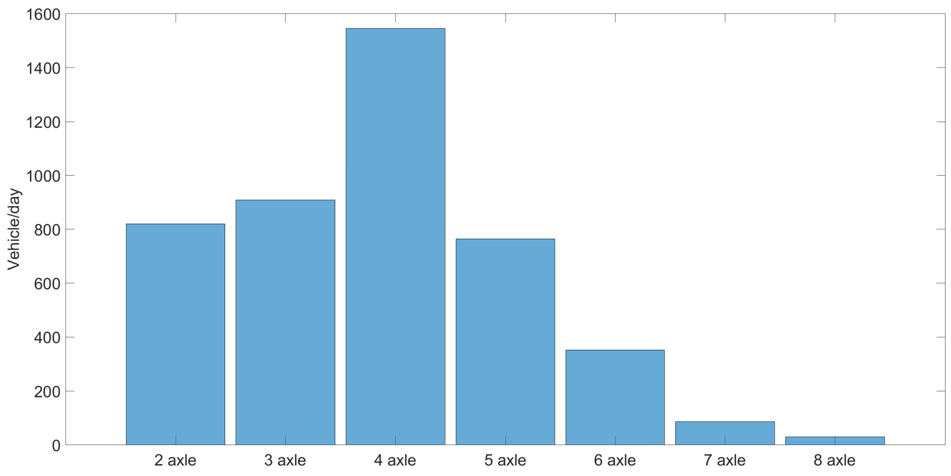


Figure 6.2: Axle count for 4 January 2016

The second way to describe what type of vehicles cross the bridge, is by using the class distribution shown in figure 3.5. A small algorithm places every single truck into a category according to its attributes, based on axle count and axle spacing. The result of this can be found in figure 6.3 and figure 6.4. It can be easily seen that the most prominent truck is in class 8, which represents the four or less axle single-trailer. The close runner-up would be class 3 and 9, these are the four tire single unit and the 5-axle tractor semi-trailer respectively. Class 3 consist mostly of heavy van's, buses and campers. Class 9 consists of the somewhat larger trucks. The high amount of class 12 vehicles was not expected, and is way higher then would be deemed logical. One would expect a downward trend regarding the size of the vehicles, not upward. After inspection it was seen that some class 12 vehicles were classified that way because of faulty peak detection. The algorithm would detect peaks which are notably smaller then the peaks which supposedly represent the axles.

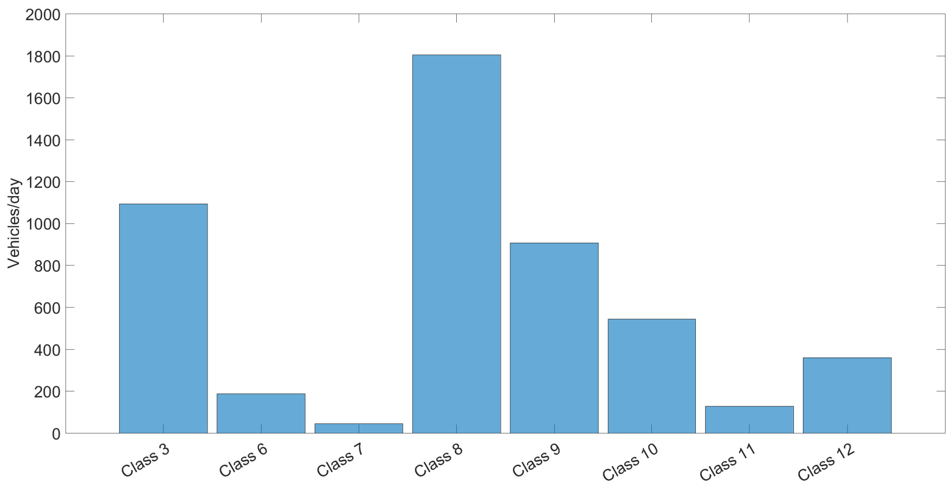


Figure 6.3: Axle count for 3 August 2015

Figure 6.5 shown a visualization of the error that sometimes occurs. One can see that visually it is highly unlikely that the second detected peak belongs to an axle. This reasoning is based on the fact that the peak is only a fraction of the height compared to the other surrounding peaks. The reason that the peak is still detected is because it has a height that marginally transcends the minimum peak height constraint. The problem with such errors, is that they are hard to detect. In this case study the data was only visually inspected, which is time consuming and not

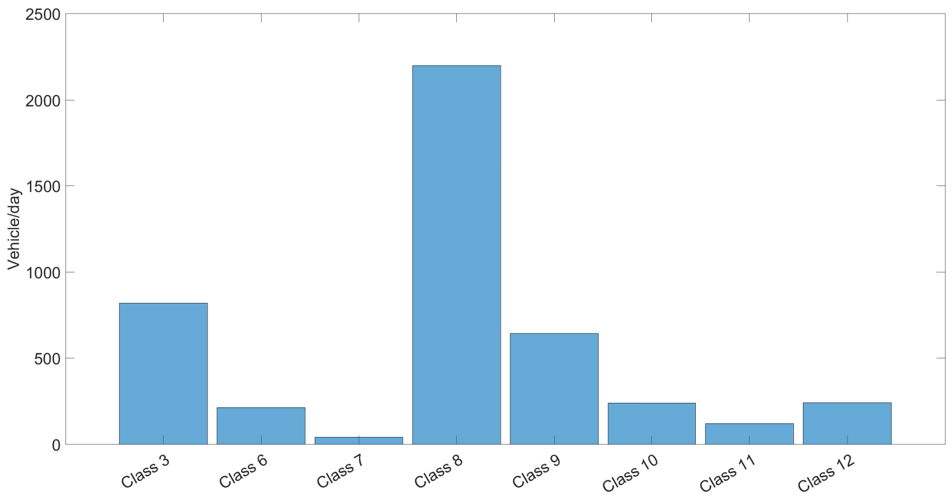


Figure 6.4: Axle count for 4 January 2016

suitable for detecting error trends. The result of this error is that a certain percentage of vehicles is placed in a different category of vehicles which it does not belong to. After inspection of the busiest 5 minutes of a day, it appeared that 7 occurrences of misinterpreted peaks occurred out of the 172 registered peaks which belonged to 41 trucks. In this inspection it was noticed that errors were occurring mostly during highly volatile strain measurements of suspected larger or heavier vehicles. As a solution one could propose to increase the constraints for peak detection. This could however cause relevant peaks to not be detected, one could also include a constraint that a detected peak cannot be a significant amount lower then the other peaks.

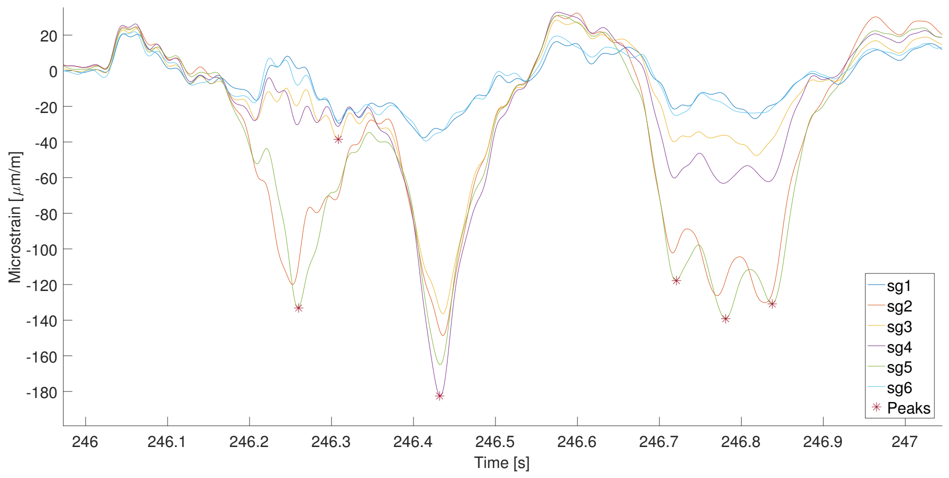


Figure 6.5: Visualization of peak detection error

6.2 Traffic intensity

Next to specific classification of vehicles, general statistics about heavy vehicles can also be plotted. Figure 6.6 shows an overview of all heavy traffic during 3 August. It consists of small bars which display the vehicle count during 5 minutes. As expected there is a small amount of traffic during the night hours, while traffic really picks up from around 6 o'clock. An interesting thing to note is that during rush hour around 8 o'clock there is a small dip in heavy traffic. This might be the transport sector trying to avoid driving in rush hour. Figure 6.7 shows another example of such a bar chart, but now with the data from 4 January. One can see a somewhat lower traffic intensity on the 4^{th} of January, this could be due to the fact that there were weather alarms during this period of time because of frosted roads [4]. There is also a moment where supposedly no heavy vehicles passed in the middle of the day. After inspection it was seen that the data was probably corrupted in this time-frame, possibly due to the freezing temperature outside.

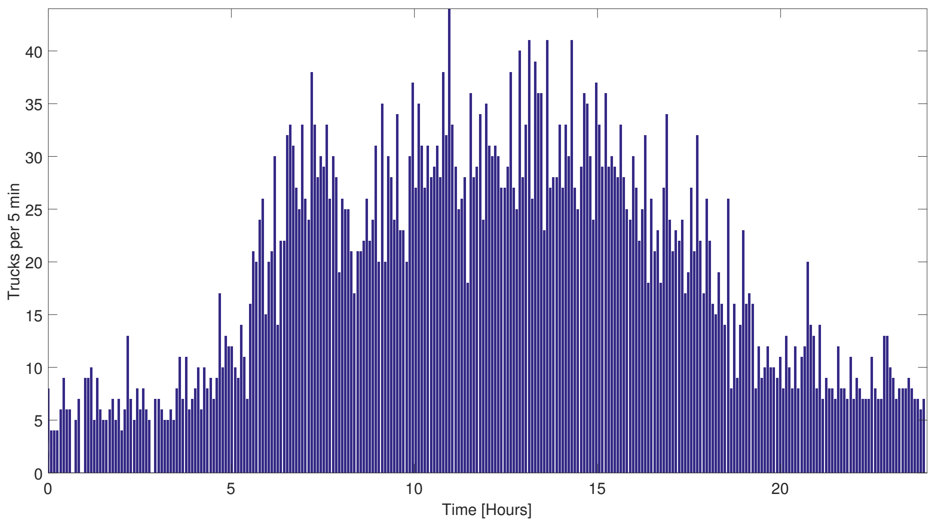


Figure 6.6: Visualization general traffic statistics during 3 August 2015

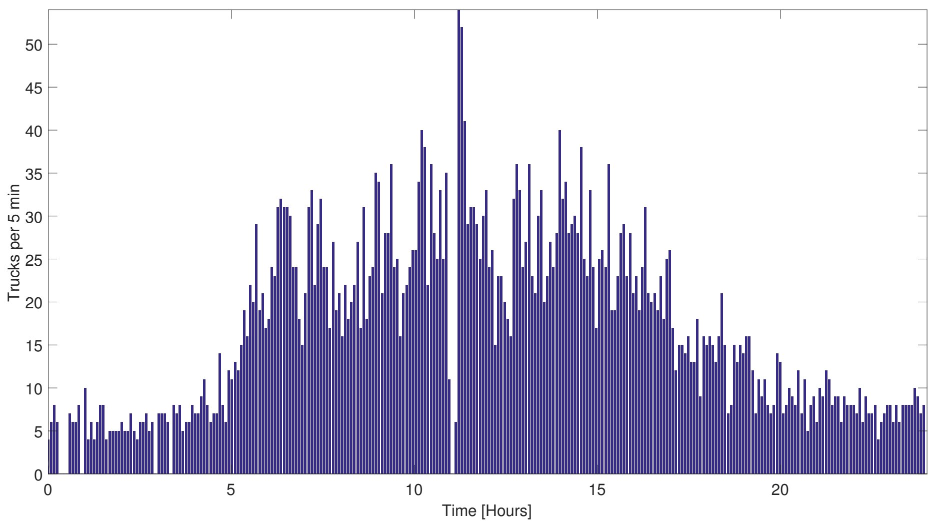


Figure 6.7: Visualization general traffic statistics during 4 January 2016

When combining the visualization of the previous histogram and bar chart, we can create an overview of which types of vehicle types traverse the bridge at which point in the day. This is shown in figure 6.8 and figure 6.9. They show the same chart from figure 6.6 and figure 6.7, but then colored with the different type of vehicles.

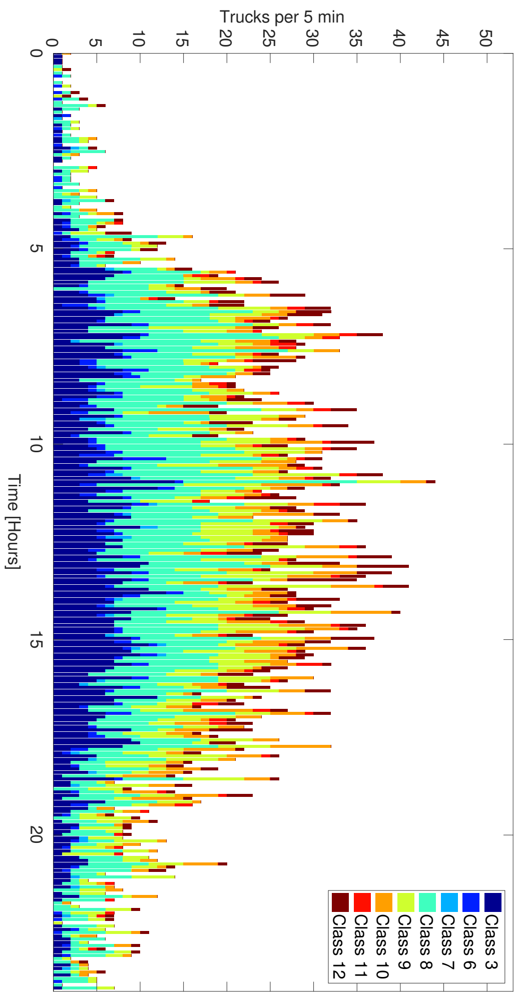


Figure 6.8: Vehicle types per 5 minutes on 3 August 2015

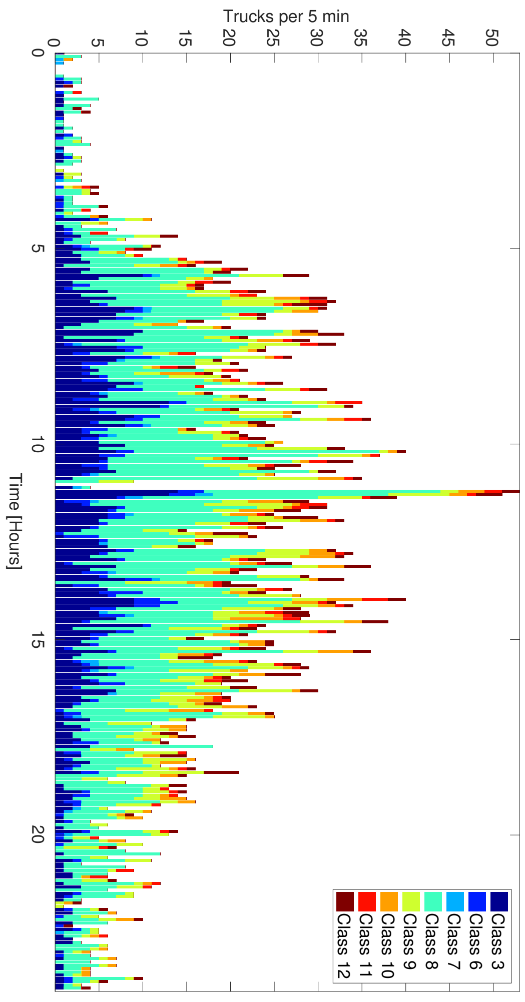


Figure 6.9: Vehicle types per 5 minutes on 4 January 2016

7 Discussion

This report analyzes the possibilities regarding the automation of interpretation of strain gauge sensor data. This is done in the context of a bridge with a orthotropic steel deck, namely the 'Van Brienennoordbrug'. The reason to collect all the strain gauge data was pre-established by Rijkswaterstaat and TNO. A research report was already written as to explore the possibilities of the data. The research proposed several ways to manually extract useful data from the raw dataset. This report builds on that existing knowledge to enable the automation of the strain gauge data analysis. This was seen as a possibility to make a first step to further research regarding the use of raw data for top level maintenance decision making. This research was based on the premise that the data could be automatically analyzed using the software MATLAB. An algorithm was formulated that imported, organized, analyzed and presented the data. The final result of this algorithm, is a matrix which contains all calculated parameters for each truck that crossed the bridge in a selected day. The visualization was presented in the report in the form of a case study. In this case study two days were selected and analyzed to test the algorithm. Traffic statistics from these days were presented in the form of bar charts and histograms.

The information gathered during this research accumulates to the fact that there are possibilities in automating the interpretation of strain gauge sensor data. During this research it was found that there was a high degree of variance in the data which was to be interpreted. This resulted in a lot of iterations of the code, as to reach a version of the program that could handle this high degree of variance. However it is to be noted that some unanticipated errors are still present in the results. Most of these unanticipated errors are the result of the static boundary conditions that are used to find peaks within the data. Some of them are resolved by the processing of the peaks, for example by requiring more than one sensor to detect a peak before excepting it as valid. However, this is not a foolproof way of preventing irrelevant peaks from being accepted as relevant data.

A large uncertainty factor remains in the output data, due to the fact that none of the output data could be verified. Therefore it is unknown whether the program interprets the raw data correctly. This is the case for example, with the interpretation of the peaks in data. The prominent peaks are assumed to be the result of wheels crossing the location of the sensors. Logic dictates that it is highly probable that this is true, however the results were not verified. Also some time-spacing was observed between the measuring of peaks from different sensors on the same longitudinal location. This could be the result of the sensors not being perfectly aligned. Both these issues require local inspection in order to investigate the reliability of the model.

7.1 Conclusion

The steps needed to compare the different strain gauges were found to be diverse. First, all data was filtered with a low-pass filter, whereafter all data needed to be normalized with a zero mean. Then based on the location of the sensors, the data was separated and processed separately due to different responses to the same input. After the analysis, the data was combined with a virtual handshake to confirm what data was accepted as relevant.

Using the law set by the government as well as vehicle size regulations, a baseline was identified which could track vehicle spacing based on the speed of the traffic. Several exceptions were introduced to the algorithm that could correct possible misinterpreted vehicle separations. This was necessary because the maximum axle spacing is quite close to the minimum vehicle spacing. Which could result in the merger of two vehicles within the program, interpreting the spacing between two vehicles as the space between two axles.

The parameters that were automatically extracted from the complete dataset were:

- Identification of individual vehicle timing
- Speed per vehicle

- Axle count per vehicle
- Axle spacing per vehicle
- Classification of vehicles

All the above mentioned parameters are also tracked in time, which means that the location of each type of detected vehicle can be traced back to the second when it crossed the sensors on the bridge. With the successful extraction of the above-mentioned parameters, it has been shown that the automation of vehicle statistic gathering from raw strain gauge data, is possible. Opening up the possibility to further the research of using raw strain gauge data in a maintenance decision making context.

7.2 Recommendations

A very first recommendation would be to start a verification process which could check whether the extracted results closely resemble reality. This could for example be done with the use of visual inspection from camera's which are present at the bridge. Then it could be established what the error rate of the program is, and what the cause of these errors are.

It would also be meaningful to look into the reaction of the strain gauges to temperature. Severe temperature changes could effect the data which could result in misinterpretation by the algorithm. An example would be a lower peak altitude due to lower temperature, which could lead to missing relevant peaks.

Another factor which has not yet been looked into, is very low speed of vehicles due to for example a traffic jam. Although the algorithm contains several parts which adjust for varying speeds, it has not been tested in very slow speed traffic.

Another step which could be taken is the addition of extra output parameters. These could include; location of the vehicle on the road, wheel count and estimated vehicle weight. The calculation of these parameters was already explored by Kallitsas, but contained high variance and uncertainty.

An additional action which could be taken to increase the accuracy of the program, would be to include extra sensors. Most importantly extra sensors on the lateral location where only three sensors exist currently. This could help increase the accuracy of the speed calculation of the vehicles.

A final recommendation would be to further investigate the remaining noise within the data. It can be seen that there are higher frequency signals still present after filtering. This is probably a dynamic response due to moving vehicles across the bridge. It might even be possible to use this to extract even more valuable data.

Bibliography

[1] Foppe Bouk Peter de JONG. *Renovation techniques for fatigue cracked orthotropic steel bridge decks*. PhD thesis, The school of the thesis, 2007.

[2] Road Safety Authority Ireland. Guidelines on maximum weights and dimensions of mechanically propelled vehicles and trailers, including manoeuvrability criteria. 2015.

[3] Panagiotis Kallitsas. Fatigue loads identification on orthotropic steel decks learning from strain measurements in practice. Master's thesis, TU Delft, 2016.

[4] KNMI. Ijzel begin januari 2016. 2016.

[5] Dooren van F. & Kolstein M. H. Maljaars, J. Fatigue assessment for deck plates in orthotropic bridge decks. *Steel Construction - Design and Research*, 5(2):93–100, 2012.

[6] Mathworks. *findpeaks: Find local maxima*, 2017. https://nl.mathworks.com/help/signal/ref/findpeaks.html.

[7] U.S. Department of Transportation. Traffic monitoring guide. 2016.

[8] Dienst Wegverkeer. Overzicht maten en gewichten in nederland. 2012.

A Sensor layout

Table A.1: Detailed sensor coordinates on the bridge

sensor	x [mm]	y [mm]	z [mm]
sg1	-4508	-1200	-331
sg2	-4508	-1800	-331
sg3	-4508	-2400	-331
sg4	-4508	-3600	-331
sg5	-4508	-4200	-331
sg6	-4508	-4800	-331
sg7	-4508	-1383	-6
sg8	-4508	-1617	-6
sg9	-4508	-1983	-6
sg10	-4508	-2217	-6
sg11	-1800	-1500	-6
sg12	-1800	-1383	-6
sg13	-1800	-1617	-6
sg14	-1800	-1800	-331
sg15	-1800	-1983	-6
sg16	-1800	-2217	-6
sg17	-1800	-2100	-6
sg18	-560	-1500	-6
sg19	-560	-1800	-331
sg20	-560	-1617	-6
sg21	-560	-1500	-6
sg22	-560	-1983	-6
sg23	-560	-2217	-6
sg24	-560	-2100	-6
sg25	-4508	-1500	-6
sg26	-3025	-1500	-6
sg27	-3775	-1800	-331
sg28	-3025	-1800	-331
sg29	-2225	-1800	-331
sg30	-1375	-1800	-331
sg31	-4508	-2100	-6
sg32	-3025	-2100	-6

B Peak detection

Algorithm B.1: MATLAB code to find peaks

```matlab
%eF is a cell that contains all filtered normalized sensor signals.
%Findpeaks is executed to find and store all peaks with certain boundary
    conditions.
for i = 1:length(eF{j}(1,:))
    [height{i},locs{i},~,p{i}] = findpeaks(-eF{j}(:,i),t{j},'minpeakheight',25,'
        MinPeakProminence',10,'MinPeakDistance',0.05);
end
```

C Speed estimation

Algorithm C.1: MATLAB code to estimate speed over 5 minutes

```matlab
%determine estimated speed over 5 minutes
tmp_peak = (sensor2{j}(:,2) > 80).*sensor2{j};  %detect most prevalent peaks
tmp_peak(~any(tmp_peak,2),:) = [];              %remove empty rows
tmp_peak2 = (sensor27{j}(:,2) > 80).*sensor27{j};
tmp_peak2(~any(tmp_peak2,2),:) = [];
if isempty(tmp_peak) || isempty(tmp_peak2)
    %if no high peak is detected, assume speed as 80 km/h
    speed(j) = 80/3.6;
else
    %detect which peaks belong to each other, then  calculate what the
    %time between them is.
    inter = tmp_peak(:,1) - tmp_peak2(:,1)';
    %negative values are not relevant
    inter(inter<0) = inf;
    %find closest positive peak pairs
    for i=1:length(inter(1,:))
        inter2(i) = min(inter(:,i));
    end
    inter = inter2;
    inter(inter==0) = [];
    %convert time between peaks to m/s with distance between sensors
    inter(1,:) = 0.734./inter(1,:);
    speed(j) = sum(inter)/length(inter);    %take average of speed over 5 mins
end
```

D Grouping peaks

Algorithm D.1: MATLAB code group peaks

```matlab
%merge all the sensor data in one matrix, sort this matrix based on
%time of located peak.
merged = vertcat([sensor1{j}],[sensor2{j}],[sensor3{j}],[sensor4{j}],[sensor5{j
    }],[sensor6{j}],[sensor25{j}],[sensor31{j}],[sensor7{j}],[sensor8{j}],[sensor9
    {j}],[sensor10{j}]);
sorted = sortrows(merged);

peaks = 1;
i = 1;
while i < length(sorted(:,1))
    %find all rows after row i that fall within the interval 1/speed,
    [row,~] = find(sorted(:,1) >= sorted(i,1) & sorted(:,1) <= (sorted(i,1) + 1/
        speed(j)));
    overlapping_peaks = length(row);
    if overlapping_peaks == 1
        i = i + overlapping_peaks;  %if only 1 sensor detects a peak, ignore it
        continue
    end
    %merge all detected peaks into the most prominent peak and store
    %that in a confirmed peak matrix
    [confirmed_peaks(peaks,2), indx] = max(sorted(i:(i+overlapping_peaks-1),2));
    confirmed_peaks(peaks,1) = sorted(i+indx-1,1);
    confirmed_peaks(peaks,3) = overlapping_peaks;

    %If after initially confirming the peaks there is difference
    %between 2 peaks of less then 1/speed, merge the peaks.
    if peaks > 1
        if (confirmed_peaks(peaks,1) - confirmed_peaks(peaks - 1,1)) < 1/speed(j)
            if (confirmed_peaks(peaks,1) - confirmed_peaks(peaks - 1,1)) < 0
                peaks = peaks - 1;
            else
                confirmed_peaks(peaks - 1,:) = confirmed_peaks(peaks,:);
                confirmed_peaks(peaks,:) = [];
                peaks = peaks -1;
            end
        end
    end
    peaks = peaks + 1;
    i = i + overlapping_peaks;
end
%check if any confirmed peaks are found, if so store them in cell array
if exist('confirmed_peaks')
    validated_peaks{j} = confirmed_peaks;
else
     validated_peaks{j} = [];
end
```

E Vehicle identification

Algorithm E.1: MATLAB code for identifying vehicles

```
counter = 0;
axle = 0;
%determine which peaks belong to which vehicle
if isempty(validated_peaks{j})
    %there are no trucks in this timeslot
else
    if length(validated_peaks{j}(:,1)) > 1
        for k = 2:length(validated_peaks{j}(:,1))
            temp = validated_peaks{j}(k,1) - validated_peaks{j}(k-1,1);
            %compare time between peaks with allowed time between
            %vq is the abovementioned table
            if (temp > vq(round(3.6*speed(j)))) || (k == length(validated_peaks{j
                }(:,1)))
                counter = counter + 1;
                trucks{j}(counter,2) = validated_peaks{j}(k-axle,1);
                if k == length(validated_peaks{j}(:,1))
                %Take the last iteration into account in each dataset
                    if axle == 0
                        axle = axle + 2;
                        trucks{j}(counter,2) = validated_peaks{j}(k+1-axle,1);
                    else
                        axle = axle + 1;
                    end
                end
                trucks{j}(counter,1) = axle;
                axle = 1;
            else
                axle = axle + 1;
            end
        end
    end
end
```

F Speed and axle spacing

Algorithm F.1: MATLAB code for calculating speed and axle spacing

```
for i=1:length(trucks{j}(:,1))
    if trucks{j}(i,1) > 1
        temp = trucks{j}(i,2) - sensor27{j}(:,1);
        temp1 = trucks{j}(i,2) - sensor28{j}(:,1);
        temp2 = trucks{j}(i,2) - sensor29{j}(:,1);
        if all(temp <= 0) == 1
            %velocity(1) = [];
        else
            velocity(1) = 0.734./min(temp(temp > 0));
        end
        if all(temp1 <= 0) == 1
            %velocity(2) = [];
        else
            velocity(2) = 1.484./min(temp1(temp1 > 0));
        end
        if all(temp2 <= 0) == 1
        else
            velocity(3) = 2.284./min(temp2(temp2 > 0));
        end
        if exist('velocity')
            velocity = velocity(velocity > max(velocity)/2); %filter out wrong
                peak correlations
            trucks{j}(i,3) = mean(velocity);
        else
            trucks{j}(i,3) = 0;
        end

        clear temp temp1 temp2
        temp = zeros(1,trucks{j}(i,1));
        %find all peaks that belong to each truck
        [loca, locb] = ismember(trucks{j}(i,2),validated_peaks{j});
        for k=0:(trucks{j}(i,1)-1)
            temp(1,k+1) = validated_peaks{j}(locb+k,1);
        end

        %calculate the distance between each axle and store them
        for k=1:(length(temp)-1)
            trucks{j}(i,3+k) = (temp(k+1)-temp(k))*trucks{j}(i,3);
        end

    end
end
```

G Truck corrections

Algorithm G.1: MATLAB code for splitting wrongfully merged axles

```matlab
[row, col] = find(trucks{j}(:,4:end)>10);
if not(isempty(col))
    col = col + 3;
    temp = horzcat(row,col);
    temp = sortrows(temp,-1);
    row = temp(:,1);
    col = temp(:,2);
    for k = 1:length(row)
        if trucks{j}(row(k),col(k)) == trucks{j}(row(k),3-1+trucks{j}(row(k),1))
            %no new vehicle, just 1 separate peak
            trucks{j}(row(k),col(k)) = 0;
            trucks{j}(row(k),1) = trucks{j}(row(k),1) - 1;
        else
            trucks{j} = [trucks{j}(1:row(k),:);zeros(1,length(trucks{j}(i,:)));
                trucks{j}(row(k)+1:end,:)];
            trucks{j}(row(k)+1,4:3+length(trucks{j}(row(k),col(k)+1:end))) =
                trucks{j}(row(k),col(k)+1:end);
            trucks{j}(row(k)+1,1) = (3+trucks{j}(row(k),1)-1) - col(k) + 1; %axle
                count new vehicle
            trucks{j}(row(k),1) = (col(k)-4+1);     %correct first vehicle axle
                count
            trucks{j}(row(k),col(k):end) = 0;       %remove axles from first
                vehicles
            trucks{j}(row(k)+1,3) = trucks{j}(row(k),3); %set speed new vehicle
            [loca, locb] = ismember(trucks{j}(row(k),2),validated_peaks{j});
            trucks{j}(row(k)+1,2) = validated_peaks{j}(locb+col(k)-3,1); %find
                where vehicle starts in time
            if k < length(row)
                if row(k) == row(k+1)
                    %if 2 errors in one row, search again in new matrix
                    row(k+1) = row(k+1) + 1;
                    col(k+1) = col(k+1) - col(k) + 3;
                end
            end
        end
    end
end
```

H Raw data

Table H.1: Raw data example from 3 August 2015

Time 1 - default sample rate CH=1	**temp CH=10**	**humidity CH=11**	**T1 wegdek CH=12**	**T2 brugdek CH=13**	**strain1 CH=20**	**strain2 CH=21**	**strain3 CH=22**	**strain4 CH=23**	**strain5 CH=24**
0	26.07	63.61	19.38	19.94	10.06	13.36	18.26	25.54	28.33
0.00083	26.07	63.61	19.38	19.94	9.843	13.28	18.07	23.24	26.44
0.00167	26.07	63.61	19.38	19.94	8.538	10.97	16.78	23.24	25.62
0.00250	26.07	63.61	19.38	19.94	9.120	10.82	16.40	22.96	26.49
0.00333	26.07	63.61	19.38	19.94	9.277	13.48	17.59	22.15	26.08
0.00417	26.07	63.61	19.38	19.94	5.629	8.774	14.08	20.77	24.12
0.00500	26.07	63.61	19.38	19.94	8.223	9.725	15.94	22.86	25.96
0.00583	26.07	63.61	19.38	19.94	7.233	10.79	15.60	21.57	25.56
0.00667	26.07	63.61	19.38	19.94	7.476	10.97	16.95	22.98	25.64
0.00750	26.07	63.61	19.38	19.94	7.162	9.017	15.12	21.45	25.31
0.00833	26.07	63.61	19.38	19.94	6.069	9.033	14.48	20.49	22.71
0.00917	26.07	63.61	19.38	19.94	7.469	8.373	13.36	19.95	24.28
0.01000	26.07	63.61	19.38	19.94	3.947	9.332	15.05	19.76	24.48
0.01083	26.07	63.61	19.38	19.94	7.052	8.255	14.27	22.00	23.48
0.01167	26.07	63.61	19.39	19.94	7.052	9.851	16.33	21.45	26.20
0.01250	26.07	63.61	19.39	19.94	7.343	9.222	14.65	22.33	24.61
0.01333	26.07	63.61	19.39	19.94	5.464	11.53	15.35	21.31	25.23
0.01417	26.07	63.61	19.39	19.94	10.74	9.458	18.06	24.03	26.65