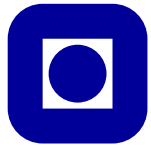


Norwegian University of Science and Technology

Master Thesis

Classification of fluids in a closed tank with temperature changes and noise

---

Håkon Søhoel

June 2, 2014

Problem description

To reduce emission of NO_x from diesel engines the method Selective catalytic reduction (SCR) is used in heavy duty commercial vehicles. In this system AdBlue is used as a reducing agent. To avoid damages to the engine caused by the use of other fluids than AdBlue it is important to detect the presence of such a fluid.

This project will look into the possibilities of detecting and classifying fluids in a tank based on sensors already present in a current system, and possibly by adding new sensors if this is advantageous. The work is a continuation of a project where classification was achieved under stable noiseless conditions without temperature changes. As a next step temperature changes in the fluids will be introduced, and more noise will be added. The main aspects of the assignment will be:

- Developing a classifier able to cope with temperature changes and noisy environment, which are conditions challenging the ability to perform a correct identification.
- Comparing what can be achieved by only using the sensors present in a current system with what can be gained by adding new sensors.

Preface

This report has been made as part of the masters thesis concluding my five years at the Norwegian University of Science and Technology studying Engineering Cybernetics.

The master thesis has been conducted in cooperation with Wema System AS, and I would like to thank Wema for their help and advice during the last few months, and for being allowed to use their laboratory in Bergen. I would also like to thank my supervisor from the Department of Engineering Cybernetics at NTNU, Tor Onshus.

In addition, I owe many thanks to Håkon Bøe, Simen Fuglaas, Erlend Kvinge Jørgensen, Øyvind Ulvin Halvorsen, Sverre Kvamme and Kristian Stormo, the guys I've been sharing an office with during the last five months, for many rewarding discussions and for making this time more memorable. Last but not least I will thank my girlfriend, Meike Jensen, for her help and encouragement throughout the work with this thesis.

Trondheim, May 2014
Håkon Søhoel

Summary & Conclusion

In order to meet the increasingly stringent emission standards for heavy duty commercial vehicles world wide, systems to reduce the emission of NO_x from diesel engines are required. *Selective catalytic reduction*, using Adblue as a reducing agent, is such a system. To avoid damage to the system it is important to avoid non compliant Adblue being present in the tank.

This report investigates whether it is possible to distinguish different fluids from each other by applying classification theory to the sensor data available in a system for measuring urea concentration already in use. Sensor data has been retrieved by measuring in a number of fluids in a lab, while trying to simulate realistic tank conditions in the form of bubbles and temperature changes. The sensors used were an ultrasound sensor and a conductivity sensor. The fluids included in the experiment were Adblue 32.1 %, Adblue 15.8 %, Diesel, Glycol 100 %, Glycol 50 %, and water.

Three classification strategies were tested, Minimum error classification assuming Gaussian distribution of data, k-nearest-neighbor classification and least mean square classification. The best results were achieved after transforming the feature space by applying Multiple Discriminant Analysis. Using this as a preprocessing step the best results achieved by the different classification strategies were as follows: The minimum error rate (MER) classifier achieved an error rate of 0.57%. The k-Nearest-neighbor

(kNN) classifier achieved an error rate of 0.12%. The Least-mean-square (LMS) classifier achieved an error rate of 5.72%. By these results it can be concluded that the minimum error rate classifier is the recommended classification strategies of the three that were tested. This can be done because the results achieved by this classifier are almost as good as the results achieved by the kNN classifier, while after it has been trained it has a significantly lower computational cost than kNN, which makes it more suited to be implemented in a microcontroller system operating on a vehicle.

Contents

List of Figures

List of Tables

List of Program Code

Chapter 1

Introduction

In order to meet the increasingly stringent emission standards for heavy duty commercial vehicles world wide, systems to reduce the emission of NO_x from diesel engines are required. *Selective catalytic reduction*, using Adblue as a reducing agent, is such a system. Adblue is a water solution ideally containing 32.5 % urea. To avoid damage to the system it is important to avoid non compliant Adblue being present in the tank. On the market today sensor solutions measuring the urea concentration in the tank exists. However, their ability to recognize non Adblue solutions are limited, and some fluid compositions might be wrongly perceived as being Adblue.

This project seeks to apply known classification theory, in order to better distinguish different fluids that might be present in the tank, either intentionally or by human error. The work being presented in this report is the continuation of an earlier project where a system, successfully distinguishing between a selection of fluids under stable conditions in room temperature, was made [26]. In this report the development of a classifier, also able to deal with temperature changes that will occur during normal operation on a vehicle, will be presented. The classifier will be

based on data from sensors already available in a system measuring urea concentration.

The report starts with a Theory chapter presenting the mathematical basis for the classifiers, this chapter has been taken from the project work this master thesis builds upon. The work having been done as part of this master thesis is presented in three main chapters. Chapter 3 describes the process of retrieving sensor data in the laboratory and extracting features to be used for classification. Chapter 4 presents the development of a new round trip time detection algorithm and the motivation behind it. Finally, Chapter 5 describes the development and evaluation of three classifiers and the results they produced. Discussion and suggestions for future work has been placed in the end of the report. Appendix A contains output from systematic testing of the four classifiers that were developed and tested in the process. Appendix B contains paths to files referred to in the report.

Chapter 2

Theory on Classification

In this chapter some of the mathematical concepts used in this report will be briefly explained. The chapter has been taken from the report of the project work this master thesis builds upon [26] as the theory behind the classification algorithms still remains the same.

A classification system can be seen as the task of recognizing patterns in sensor data making it possible to separate the data into classes. Such a system typically consists of sensing, processing the sensor data, and classification based on the measurement [9]. An illustration of such a system can be seen in Figure 2.1. This chapter will focus on the classification module.

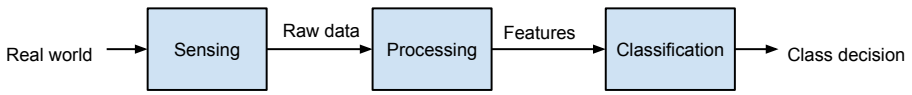


Figure 2.1: Components in a typical classification system

Many different strategies can be used within the field of classification. However a certain standard syntax on how to formulate the problem exists. A vector $\boldsymbol{x}$ contains all the measured features, and a class is defined as ω. A decision rule is made to decide which class the vector $\boldsymbol{x}$ will be assigned to. This can be done by creating a discriminant function $g(\boldsymbol{x})$ returning a scalar number, and then make a decision based on a threshold value. An example of such a classification problem is as follows. Given a problem with c classes the discriminant function can be made as a set of discriminant functions $g_i(\boldsymbol{x})$, $i = 1, .., c$. A decision rule is then typical made as: Assign feature vector $\boldsymbol{x}$ to class ω_i if $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$ for all $j \neq i$ [10]. The task of making the classifier is now reduced to finding the functions $g_i(\boldsymbol{x})$ minimizing the number of erroneous classifications. In this report three methods for finding such functions will be used and the theory behind each method will now be presented.

2.1 Nearest-neighbor

The Nearest-Neighbor classification is based on imagining all the feature vectors $\boldsymbol{x}$ as points in a n-dimensional euclidean space. A training set is made by making a number of measurements with known classes. Classification of new measurements with unknown class is then achieved by calculating the distance between the new measurement $\boldsymbol{x}$ to all the points in the training set and assigning a class to the measurement corresponding to the class of the nearest point in the training sett, hence the name Nearest-Neighbor classification. A two dimensional example can be seen in Figure 2.2. The main advantages of the Nearest-Neighbor classifier are that it is very easy to implement and that no previous knowledge of the nature of the classes is demanded. However the computational cost grows with the number of elements in the training set and the number of dimensions in the feature space. This method is therefore primarily useful as

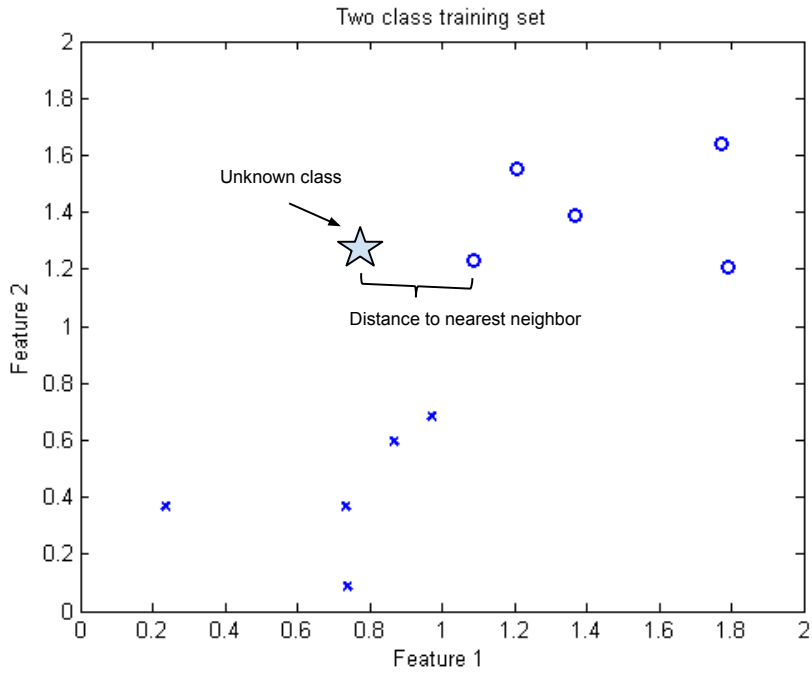


Figure 2.2: Example of nearest-neighbor classification with a two dimentional feature space and two classes. Class 1 is ploted as x and class 2 is ploted as o. The unknown feature vector ploted as a star will be assigned to class 2.

part of the development phase, where its performance can be compared to other classification strategies.

2.2 Bayesian, Minimum-error-rate

The Minimum-error-rate classifier is based on the assumption that we can define probability density functions $p(\boldsymbol{x}|\omega_i)$ for each class ω_i, $i = 1, .., c$. Using Bayes formula [7]:

$$P(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)P(\omega_i)}{p(\boldsymbol{x})} \tag{2.1}$$

we get an expression for the probability of a certain class given the information obtained in the feature vector $\boldsymbol{x}$. Here $P(\omega_i|\boldsymbol{x})$ is the probability of the class ω_i given the measurement obtained in the feature vector $\boldsymbol{x}$. $P(\omega_i)$ is the probability of having class ω_i prior to the measurement of its features, $p(\boldsymbol{x}|\omega_i)$ is the probability density function of $\boldsymbol{x}$ given it belongs to class ω_i and $p(\boldsymbol{x})$ is the unconditional probability density function of $\boldsymbol{x}$. A set of discriminant functions can now be defined as $g_i(\boldsymbol{x}) = P(\omega_i|\boldsymbol{x})$, $i = 1, .., c$. The decision rule can be stated in the standardized way: Assign feature vector $\boldsymbol{x}$ to class ω_i if $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$ for all $j \neq i$. In other words always choose the class with the highest probability given the measured features. The advantages of this classifier are that given correct knowledge of the probability density functions it is simple to implement and it is, in contrast to the Nearest-Neighbor classifier, computationally cheap when implemented in code. The main challenge is to obtain correct knowledge about which probability distribution to use and given this, estimate the correct parameters.

In the following multivariate Gaussian density is assumed. A set of discriminant functions has been defined as $g_i(\boldsymbol{x}) = P(\omega_i|\boldsymbol{x})$, $i = 1, .., c$. Since all the function are divided by the same factor $p(\boldsymbol{x})$ this term can be omitted without altering the outcome of the decision rule. Thus a new set of discriminant functions can been defined as $g_i^*(\boldsymbol{x}) = p(\boldsymbol{x}|\omega_i)P(\omega_i)$. By using the property $a > b => ln(a) > ln(b)$ the discriminant functions can be further altered to the set $g_i^{**}(\boldsymbol{x}) = ln(p(\boldsymbol{x}|\omega_i)P(\omega_i))$, again without altering the outcome of the decision rule. Inserting $p(\boldsymbol{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ we have the set of functions:

$$g_i^{**}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^{\boldsymbol{T}}\boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)-\frac{d}{2}ln(2\pi)-\frac{1}{2}ln(|\boldsymbol{\Sigma}_i|)+ln(P(\omega_i)), i = 1,..c \tag{2.2}$$

In Equation 2.2 $\boldsymbol{\mu}_i$ is the expected feature vector given class ω_i, $\boldsymbol{\Sigma}_i$ is the covariance matrix of the probability density function $p(\boldsymbol{x}|\omega_i)$, and d is the dimension of the feature space. The term $-\frac{d}{2}ln(2\pi)$ can be removed because it is equal for all $g_i^{**}(\boldsymbol{x})$. Finally, by sorting terms after decreasing order, we end up with a set of discriminant functions [10]:

$$g_i(\boldsymbol{x}) = \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{w}_i^T\boldsymbol{x} + w_{i0} \tag{2.3}$$

$$\boldsymbol{W}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1} \tag{2.4}$$

$$\boldsymbol{w}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i \tag{2.5}$$

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^{\boldsymbol{T}}\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}ln(|\boldsymbol{\Sigma}_i|) + ln(P(\omega_i)) \tag{2.6}$$

The parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$, and $P(\omega_i)$ can be estimated for each class by using a training set containing measurements with known class. Using these estimated parameters $\boldsymbol{W}_i$, $\boldsymbol{w}_i$ and w_{i0} can be calculated directly. To classify a new measurement $\boldsymbol{x}$ with unknown class all that needs to be done is calculate the value of $g_i(\boldsymbol{x})$ for all $i = 1,..c$ and assign ω_i to $\boldsymbol{x}$ if $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$ for all $j \neq i$.

2.3 Least-mean-square

The least-mean-square method seek a minimum error solution to the equations:

$$\begin{aligned} \boldsymbol{a}_i^{\boldsymbol{T}}\boldsymbol{y} &= 1 \ \ \forall \boldsymbol{y} \ \in \ Y_i \\ \boldsymbol{a}_i^{\boldsymbol{T}}\boldsymbol{y} &= 0 \ \ \forall \boldsymbol{y} \ \notin \ Y_i \end{aligned} \tag{2.7}$$

In equation 2.7 $\boldsymbol{y}$ is an expanded version of $\boldsymbol{x}$:

$$\boldsymbol{y} = \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} \tag{2.8}$$

and Y_i is the set containing all $\boldsymbol{y}$ belonging to class ω_i

When having multiple classes the following syntax can be used:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \boldsymbol{Y}_2 \\ \vdots \\ \boldsymbol{Y}_c \end{bmatrix} \tag{2.9}$$

where all the vectors $\boldsymbol{y}$ belonging to class ω_i fills all the rows in $\boldsymbol{Y}_i$.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_c \end{bmatrix} \tag{2.10}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_1 \\ \boldsymbol{B}_2 \\ \vdots \\ \boldsymbol{B}_c \end{bmatrix} \tag{2.11}$$

where all the elements in $\boldsymbol{B}_i$ are zero except the ones in the ith column, those are 1. Ideally a matrix $\boldsymbol{A}$ should be found so that the equation $\boldsymbol{Y}\boldsymbol{A} = \boldsymbol{B}$ holds. This will usually not be possible, and instead a matrix

$\boldsymbol{A}$ minimizing the square error of the equation $\boldsymbol{YA} - \boldsymbol{B}$ is found. A can then be found as [11]:

$$\boldsymbol{A} = (\boldsymbol{Y^T Y})^{-1} \boldsymbol{Y^T B} \tag{2.12}$$

Chapter 3

Retrieving data & extracting features

The goal of this project is to be able to classify fluids in a tank using data retrieved from sensors. The final system is meant to be part of a real time system continuously reporting the content in the tank. As a result the sensors used needs to work in a closed tank with little or no need of supervision and maintenance. At the current stage of the study it is important to define what kind of sensors can be placed in the tank, and which features can be extracted from the available sensor data. Seeing that the tank will be placed on a vehicle, important issues are the limited space available, the harsh conditions the system must endure and the cost of the system. The work concerning development and testing of new and already existing sensors has been performed in a lab. Conditions the finished system will meet on a vehicle have been reproduced to the extent possible in the lab. Creating the laboratory setup has consisted of creating a LabVIEW program combining many sensors into one system. Some hardware implementations have also been necessary in order to combine new circuits and already existing hardware. In addition to creating a laboratory setup

to retrieve test data, algorithms for feature extraction have been created in Matlab. This chapter will start by describing the measurement setup creating data files containing sensor data before it defines the features to be extracted and the program performing the feature extraction.

3.1 Measurement setup

The measurement setup consists of a tank containing a fluid, a heating device to create the change of temperature, and the sensors together with a LabVIEW program running on a computer for data acquisition. This part of the system can be seen as the sensing module in Figure 2.1. The LabView setup collects data including an ultrasound signal, temperature, conductivity, and light absorption at five different light frequencies (wave length: 505 nm, 573 nm, 589 nm, 605 nm, 624 nm). A simplified overview of the data flow can be seen in Figure 3.1. In addition, a pump was used to create circulation and bubbles in the fluids together with a heating/cooling device to achieve a temperature sweep consistent with what a finished system will meet. In order to get as wide a temperature range as possible the fluids were cooled in a freezer over night and then heated by the heating device while performing all the measurements simultaneously. Performing one temperature sweep in a fluid typically took three hours.

3.1.1 LabVIEW program

As already stated, a LabVIEW program was made to handle all the data acquisition, see Figure 3.1. The program communicates with the the Arduino board using the *NI LabVIEW Interface for Arduino Toolkit*[21]. This consists of a program running on the Arduino and a series of LabVIEW functions making it straight forward to access the Arduino I/O pins. To retrieve the ultrasound data a National Instruments card (NI PCI-5142) was used. The part of the program sampling the ultrasound is built around

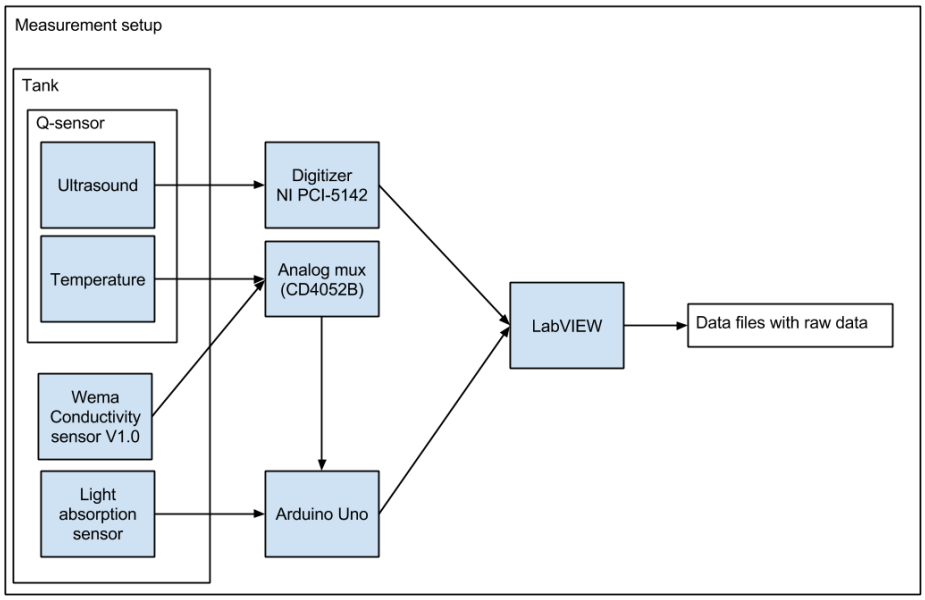


Figure 3.1: Measurement setup

an example code provided by LabVIEW.

As a result of the Arduino board only having six analog I/O pins and nine analog values needed to be read by the Arduino, an analog multiplexer (CD4052B, [8]) was used. The multiplexer was set up in such a way that it could be controlled by LabVIEW through the Arduino board. LabVIEW provides the possibility of creating laboratory applications quickly with a practical user interface. However, the graphical programing quickly results in quite untidy and unreadable block diagrams. As a result of this the LabVIEW program will be explained in writing. Still, the program file (*measuretofile.vi*) can be found by following the path in Appendix B. A typical situation during measurement in LabVIEW can be seen in Figure 3.2 showing the user interface. Three graphs can be seen, one showing the trig signal used to trigger the ultrasound transducer (top center), one

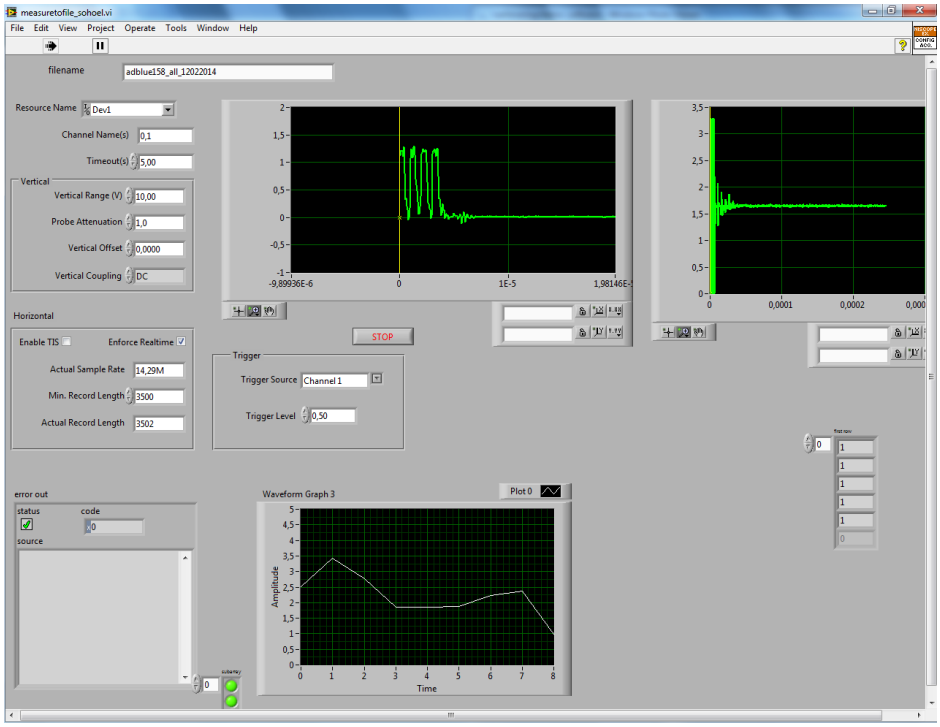


Figure 3.2: LabVIEW interface

showing the ultra sound signal (top right), and the last one showing the nine analog values being read by the Arduino board (bottom center). In addition to this, a number of text-boxes and buttons can be seen. Due to the scaling of the plot it might be difficult to read the text. However, these are mainly input values used to control the program and the important ones are the following:

- *filename*, specifying the name of the output files generated by the program. In total three files are generated using the specified filename as the root and adding respectively; *_us_signal.txt*, *_us_trig.txt* and *_arduino.txt*.

- *Resource Name*, specifying which hardware (National Instruments card) to import ultrasound data from.
- *Channel Name(s)*, specifying which of the channels available at the National Instruments card should be used. In this setup both channel 0 and channel 1 were used.
- *Timeout(s)*, specifying how long the program will wait for a trigger signal before timing out.
- *Min. Record Length*, Minimum number of samples recorded of the ultrasound signal.
- *Trigger Source*, specifying which channel is used to trigger the recording of the ultrasound signal.
- *Trigger Level*, specifying what voltage level needs to be crossed to be detected as a trigger.

When running the LabVIEW application two parallel threads are started. One thread containing a loop retrieving data from the National Instruments card (NI-loop), and one thread containing a loop controlling and retrieving data measured by the Arduino board (Arduino-loop). The two treads are synced after each iteration in their respective loops.

NI-loop

The NI-loop waits for a positive trig on the channel defined as a trigger source (should be channel 1). When this occurs it records 3500 samples from channel 0 (should be the ultrasound signal) and 100 samples from channel 1 (trig signal) at $14.29MHz$. The recorded data from channel 0 is then added to the file: *filename_ us_ signal.txt*, as a new line, and the data from channel 1 is added to the file: *filename_ us_ trig.txt*. Each data

point is separated by a *tab*. These two files will later serve as input to the Matlab application extracting features from the ultrasound signal.

Arduino-loop

The Arduino-loop reads nine analog values from the Arduino. However, the Arduino Uno SMD board being used has only six analog I/O-ports [6]. As a consequence the number of analog ports was, as already mentioned, expanded by an analog multiplexer (CD4052B, [8]). The CD4052B contains two analog 4-to-1 multiplexers controlled by two digital inputs, **A** and **B**. The pin layout and truth-table can be seen in Figure 3.3. Only the multiplexer referred to as X was used. The common X port was connected to the analog read pin 5 on the Arduino board. The digital output pins 8 and 9 on the Arduino were connected to port **A** and **B** on the multiplexer respectively. Finally, X port 0 to 2 were connected to conductivity sensor pin 2 to 4 and X port 3 was connected to the temperature pin of the Q-sensor. Analog read pin 0 to 4 on the Arduino were used to read the outputs from the light absorption sensor.

Using the connection setup described above the nine values could be read by first reading analog pin 0 to 4 on the Arduino and then by reading port 5 four times while iterating through the four possible configurations of the control signals **A** and **B**. This results in a vector containing the nine values. Said vector is then appended as a new line to the file *filename_ arduino.txt* with each data point separated by a *tab*.

3.1.2 Ultrasound & temperature sensor

The ultrasound and temperature sensor are both part of the Q-sensor currently being used to detect the Urea concentration of Adblue. The ultrasound sensor consists of a transducer and a reflector plate at a known distance from the transducer. A pulse is generated, by a circuit integrated

CD4052B			
INHIBIT	**B**	**A**	
0	0	0	0x, 0y
0	0	1	1x, 1y
0	1	0	2x, 2y
0	1	1	3x, 3y
1	X	X	None

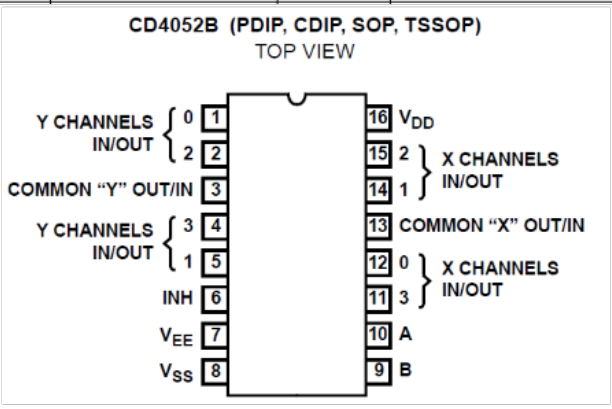


Figure 3.3: Pin layout and truth-table CD4052B. Obtained from [8].

on the Q-sensor-PCB, at fixed time intervals to excite the transducer and send a pulse through the fluid. The same transducer is then used to record the echoes returning from the reflector plate, and this signal is filtered through an active filter, again integrated in the Q-sensor-PCB. The output of this active filter is in the laboratory setup connected to channel 0 of the National Instruments card and recorded, as earlier described, by LabVIEW. The signal triggering the excitation circuit is connected to channel 1 of the National Instruments card. The temperature sensor consists of a thermistor being part of a power loop where the voltage over the thermistor

reflects the temperature. This output voltage is recorded by the Arduino board. The voltage is later translated to degrees Celsius in Matlab, based on a voltage to temperature table provided by Wema.

3.1.3 Conductivity sensor

The conductivity sensor consist of four pins mounted on a plate that can be submerged in the fluid. Pin 1 was connected to 5 volts and the three other pins were connected to resistors, with known resistance (1Ω), leading to ground. By doing so the voltage over pin 2 to 4 can be used as a measurement of the conductivity of the fluid. The value of these three pins were recorded by the Arduino.

3.1.4 Development of the light absorption sensor

In addition to the already existing sensors, measuring ultrasound and conductivity, a new sensor, measuring light absorption, has been suggested. Such a sensor is not used by the current system, but it has been successfully applied in other areas of chemical sensing [23]. A prototype of such a sensor was developed as part of a project work prior to this master [26]. In this version of the light absorption sensor a RGB-LED was used to generate light at three different frequencies. A photo resistor was then used to measure the light intensity of each frequency after the light was filtered through the fluid. Using this design, only one frequency could be read at a time. In addition the photo resistor acted quite slowly to changes in the light intensity. As a consequence of this, it was necessary to wait for about a second after changing the color of the RGB-led before measuring the light intensity at the photo resistor, resulting in a very low sampling frequency. Furthermore, the characteristics of the components were not well known, e.g. the actual wavelengths of the transmitted light was not known except for being in the red, green and blue area respectively. How-

ever, the measurement principle showed promising results and has been further studied in this report.

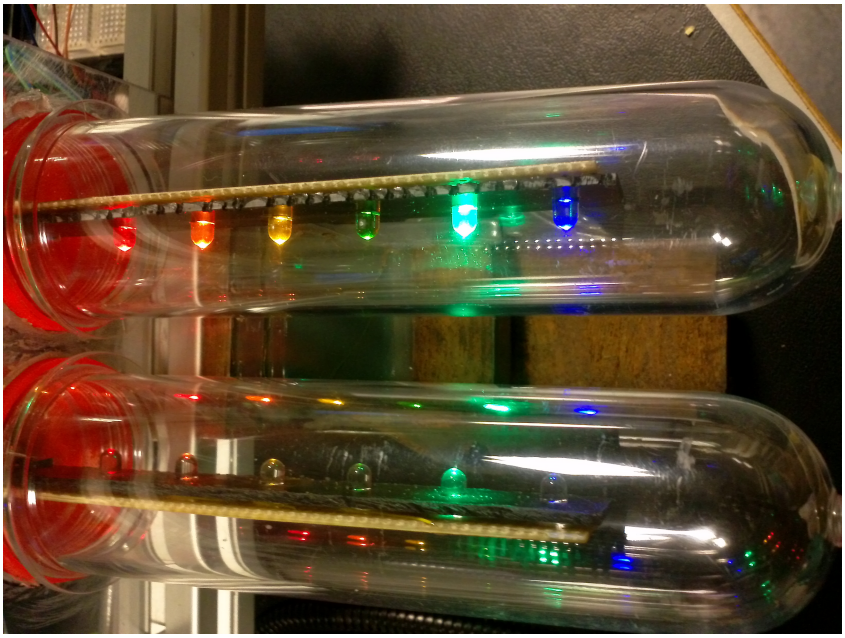


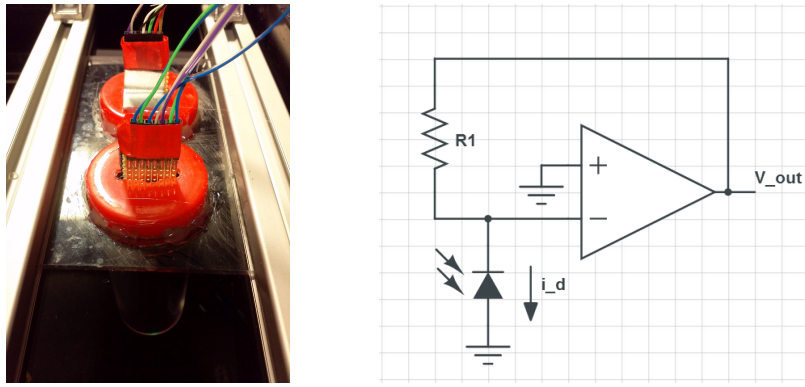
Figure 3.4: Light absorption sensor

A new light absorption sensor has been made with the goal of improving the less optimal points about the previous design. In the new design, photo diodes are used both as a light source and to measure the intensity of the light after having been filtered through the fluid. The sensor uses the principle that a light emitting diode will also generate a small electrical current when exposed to light. The current generated by the receiving diode is very small and needs to be amplified in order to be measured. A circuit was made based on a Texas Instruments tutorial [18]. The circuit is shown in Figure 3.5b. The relation between the current through the diode i_d and the output voltage v_{out} can be described by Equation 3.1.

Hence, by changing the resistor value R_1 the circuite can be tuned to give an output between zero and five volts, proportional to the current through the diode.

$$v_{out} = i_d R_1 \tag{3.1}$$

To evaluate as wide a range of frequencies as possible while still keeping the cost of the prototype at a reasonable level six LEDs were chosen. This was done on the basis of the diodes emitting light at six more or less evenly spread frequencies in the visible spectrum (wave length: 430nm [2], 505 nm [3], 573 nm [17], 589 nm [4], 605 nm [5], 624 nm [1]), and for them not being to expensive. Three ICs, from National Semiconductor (LMC6482, [19]), each containing two operational amplifiers was used to make an amplifying circuit for each LED.



(a) Part of sensor in fluid (b) Amplifying circuit based on[18]

Figure 3.5: Light absorption sensor prototype

The prototype of the sensor was built by making two rails with six LEDs on each. One working as the light source and one working as the receiver. The rails where mounted such that LEDs with equal properties

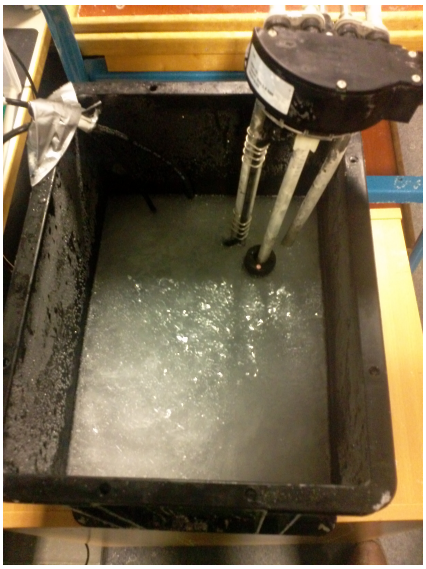
Led wavelength [nm]	Resistor value [$M\Omega$]
624	40
605	20
589	60
573	80
505	40

Table 3.1: Resistor values, light absorption sensor

where facing each other as can be seen in Figure 3.4. The LED-rails where protected by two PET tubes so they could be submerged in fluid. The amplifying circuit was mounted on a breadboard and the sensor was initially tuned by adjusting the resistor values in the circuits so that the circuit output would be close to five volts in air and zero volts if the light source was switched of. During this tuning phase no stable output was achieved by the circuit connected to the 430nm LED. As a result this LED was excluded from the further work. The remaining five LEDs all worked as intended with resistor values as described by Table 3.1. The light source rail worked by simply applying 5 volts to each of the LEDs in series with a resistor of 1 kΩ. As it was hard to perfectly tune the amplifying circuits using resistors, and because the sensor output might change slightly over time, a software calibration algorithm was created in LabVIEW. This calibration algorithm worked by simply recording the average output of the five measured values over some time while the sensor was kept in air. A factor was then found for each value to make the output in air be 1. These factors could then be used to correct the values obtained during measurements in the fluids.

3.1.5 Changing temperature

The reason to change the temperature in the fluids is to be able to compensate for the effect temperature has on the properties of the fluid. In addition to this, changing the temperature in the fluid might have an effect on the measurements in the form of noise. One example can be the formation of bubbles on every surface in the tank, this includes the sensors. These bubbles will have an impact on the measurements. Understanding the effect of such noise is important to be able to make a system that will work in the real world outside the laboratory. The fluids were cooled in a freezer over night and a heating/cooling device from *PolyScience* was used to heat the fluid while measuring, see Figure 3.6.



(a) Adblue after a night in the freezer

(b) PolyScience heating device

Figure 3.6: Cooling and heating of the fluids using a freezer and a PolyScience heating device

3.2 Features

A very important, and maybe the most difficult, part of the development of a classification system, is to decide on what features to use. A feature is information extracted from the sensor data. The choice of features to use and how to combine them is crucial in order to obtain successful classification later in the process. In the following the features investigated in this report will be presented. In addition to use features that are well known to provide relevant information about the classes of fluids, new potential features are investigated to reveal whether or not they might contain relevant information for the classification system. This part of the system can be seen as part of the processing module in Figure 2.1.

3.2.1 Speed of sound

In the system currently being used, the speed of sound in the fluid is the key feature used to both measure the urea concentration given the fluid is accepted as Adblue, and to recognize the presence of non-compliant fluids. In order to detect this feature, a pulse of sound is sent over a known distance d. The Round Trip Time, (hereby referred to as RTT) is detected. RTT is the time it takes a pulse of sound to travel back and forth between the transducer and the reflector plate. The speed of sound, v, in the fluid is then calculated as a function of the RTT and the known distance the sound has traveled, see equation 3.2.

$$v = \frac{d}{RTT} \tag{3.2}$$

3.2.2 Startup noise peak strength

In the beginning of the ultrasound signal, displayed in Figure 3.7, four peaks can be seen. These peaks are not a result of sound traveling through the fluid being measured, they are internal ringing in the transducer after

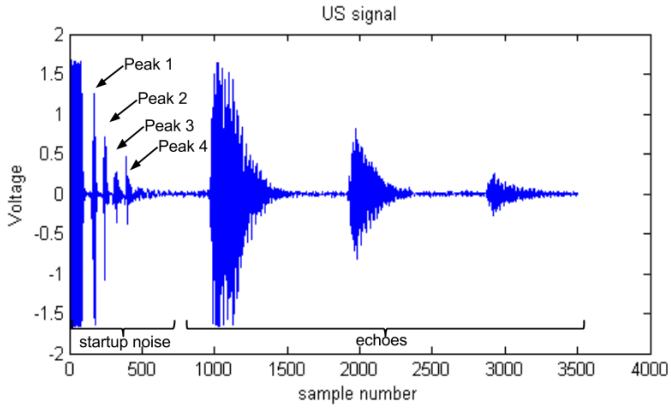


Figure 3.7: Ultrasound signal with startup noise peaks

it has been triggered. However, the acoustic impedance of the fluid might have an effect on their amplitude due to its impact on the boundary conditions between the transducer and the fluid. In order to enlighten how these peaks change with respect to the different fluids, the amplitude of these peaks has been extracted from an extensive set of ultrasound signals measured at different temperatures in different fluids. This can be seen in Figure 3.8. From the plot it seems reasonable to say that the amplitude of the peaks increases at high temperatures. In addition Adblue321 has higher values for peak 2, 3 and 4, while Glycol100 and Glycol50 have lower peak values. Although the values are similar for all classes at lower temperatures the peak amplitudes are kept as a feature to be tested with a classifier.

3.2.3 Echo amplitudes

When sound propagates through a fluid it suffers from transmission loss. This loss is due to dispersion and attenuation of the sound. Different fluids will result in different attenuation levels as a function of the shear viscos-

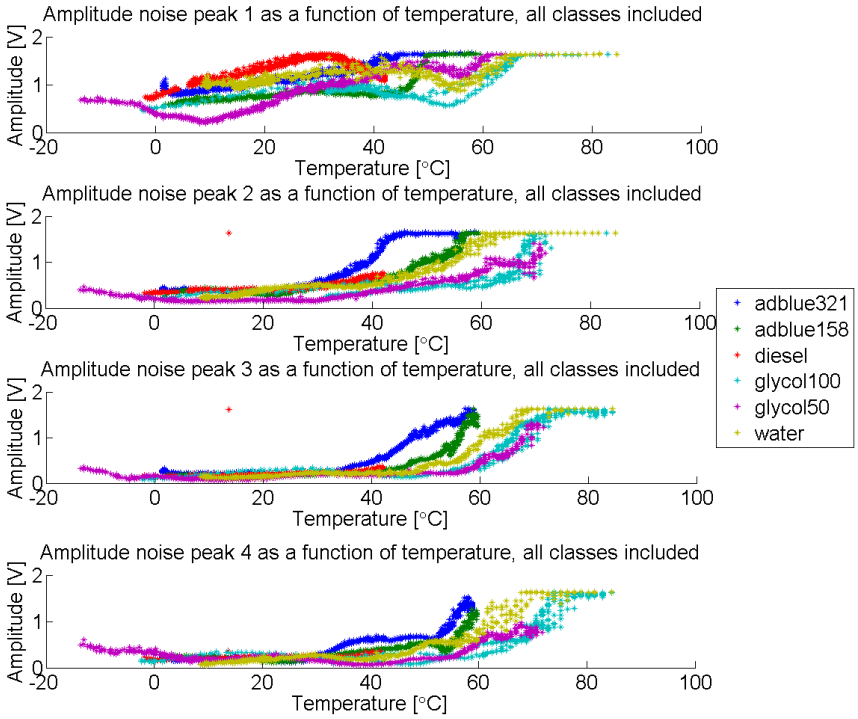


Figure 3.8: Amplitude of startup noise peaks

ity, volume viscosity, sound frequency, density and speed of sound in the fluid [14]. In addition the temperature in the fluid will have an effect on the attenuation properties. When the fluid contains bubbles, this will lead to scattering of the sound waves contributing further to the total transmission loss. As a consequence the transmission loss is affected by a number of factors, some being specific for a certain type of fluid and some being related to other factors that can be seen as noise. Whether or not the echo amplitude provides useful information about the fluid in the tank depends

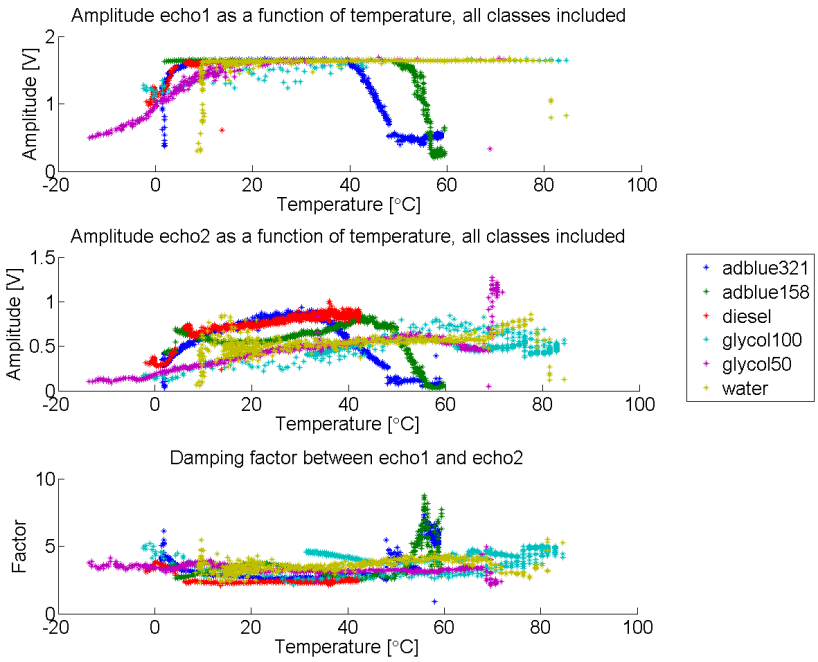


Figure 3.9: Amplitude of echo 1 (top) and echo 2 (middle) and the factor between them (bottom).

on how much noise there is hiding the potential useful information. In Figure 3.9 the amplitude of the first and second echo is shown for the measurements in different fluids at different temperatures. Furthermore, the relative factor between the two amplitudes are shown. It can be seen that the amplitude of the first echo reaches saturation at 1.65 V for most of the data points. This is due to the digital filter clipping the echo at this value, hence the actual echo has a higher amplitude. This makes it hard to use this amplitude directly as information about the real amplitude above 1.65 V is lost. However, the second echo is never strong enough to suffer from

saturation and thus still contains information about the signal strength. Because different transducers have different properties concerning signal strength measuring echo amplitude directly is not necessarily comparable between sensors. In addition, the signal strength of a transducer might change over time, and it is also affected by temperature. For this reason, to get a measurement of the transmission loss the damping factor between the first and the second echo is preferable. However, because the first echo often suffers from saturation, simply dividing the amplitude of echo 1 with the amplitude of echo 2 would result in a misleading damping factor. An algorithm was made to overcome this difficulty and provide a more correct estimate of the damping factor. This algorithm works by finding the factor a minimizing the euclidean distance between the first echo and the second echo amplified by the factor a, as described in Equation 3.3.

$$\arg\min_{a}\{\|(e_1 - ae_2)\|\} \tag{3.3}$$

Here e_1 is the first echo and e_2 is the second echo shifted in time to overlap. Figure 3.10 shows an example of how the two different methods of finding the damping factor works. At the top a plot is seen showing the second echo shifted in time to overlap with the first echo by using the already detected RTT value. In the middle the second echo has been amplified by the factor a found by using Equation 3.3. This leads to a good overlap between the two echoes, although it is not a perfect match. This is due to the distortion of the signal as it travels through the transmission channel. The bottom plot in Figure 3.10 displays the second echo on top of the first echo after it has been amplified by the factor found by dividing the amplitude of echo 1 by the amplitude of echo 2. As can be seen this leads to a perfect overlap of the echoes at the peak marked by the red circle. On the other hand the rest of the two echoes match poorly. The bottom plot in Figure 3.9 shows the damping factor found by using Equation 3.3.

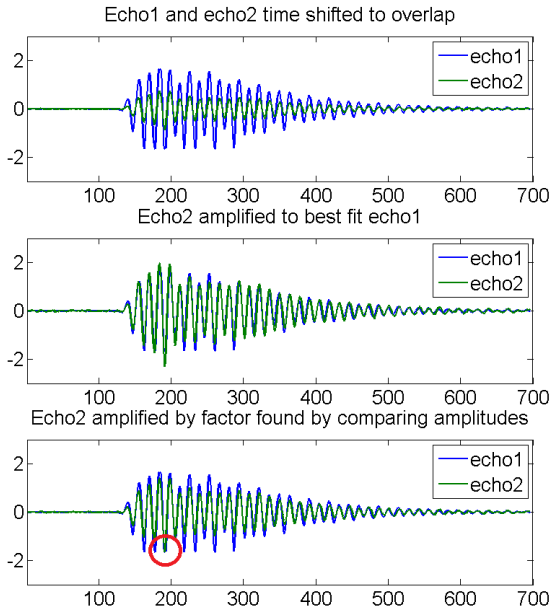


Figure 3.10: Top: Echo 2 shifted to overlap with echo 1 . Middle: Damping factor found by finding the best fit between first and second echo using Equation 3.3. Bottom: Damping factor found by dividing the amplitude of echo 1 with the amplitude of echo 2.

3.2.4 Echo frequencies

As earlier mentioned, when sound propagates through fluids it is subject to attenuation. The attenuation constant is amongst others dependent of the sound frequency. The transducer used in the ultrasound sensor has two resonance frequencies: one at about 600 KHz and one at about 1 MHz. In other words does the transmitted pulse consist of these two main frequencies. The relative damping of the two frequencies might be different depending on the fluid. To see if it might be possible to extract useful

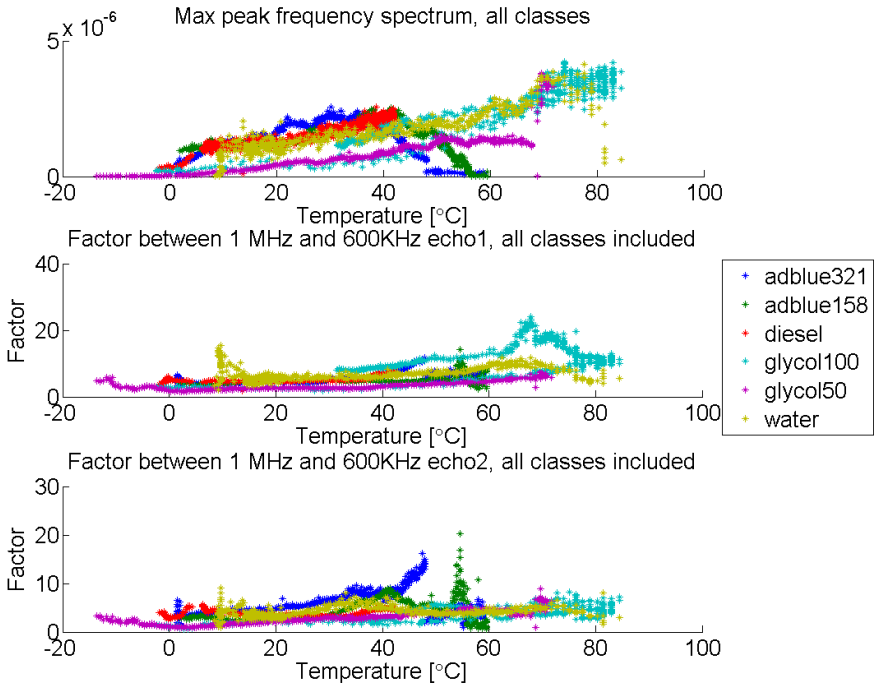


Figure 3.11: Top: Max peak frequency spectrum. Middle: Factor between 1 MHz and 600KHz echo1. Bottom: Factor between 1 MHz and 600KHz echo2.

information based on this property, the maximum peak strength of the frequency spectrum was recorded for a set of ultrasound signals measured in different fluids at different temperatures. In addition the factor between the energy at 600 KHz and the energy at 1 MHz was recorded for both the first and the second echo. All three features can be seen in Figure 3.11. The ultrasound signal is filtered through an active bandpass filter, with a pass band around 1 MHz, before it is digitized, this causes the 1 MHz frequency to be the most prominent in the recorded signal. When the fluid

contains bubbles, the frequencies in the transmitted signal will be damped depending on the resonance frequencies of the bubbles which again are related to the bubbles' radius. This means that the 600 KHz part of the signal and the 1 MHz part of the signal might be damped differently depending on the distribution of bubbles in the fluid. When the goal is to use the relative damping of the two frequencies to classify the different fluids, the fact that bubbles with different radius will damp the frequencies differently contribute to obscure the useful information. This makes it challenging to use this property as a feature for the classifier.

3.2.5 Signal variance

The variance of the ultrasound signal has been measured not because it reveals information about what fluid is present, but as a measurement of how strong the returned echoes are. When the signal variance is high this indicates that most of the signal energy is returned in the form of echoes. When the signal variance is low this indicates that much of the signal energy is absorbed in the transmission channel. This typically occurs when there are a huge amount of bubbles and has proven to be correlated with a higher fail rate in other features. The signal variance is therefore used to evaluate how trustworthy a certain measurement is.

3.2.6 Conductivity

Conductivity was, as previously, stated measured for all the classes of fluid at different temperatures. The measured values were used directly and no real preprocessing was needed. The resulting output is shown in Figure 3.12.

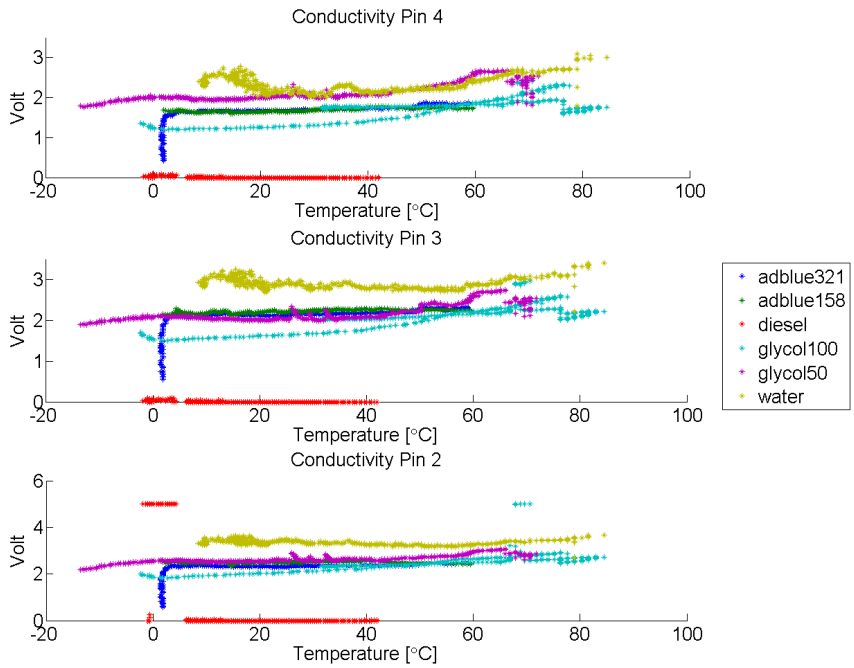
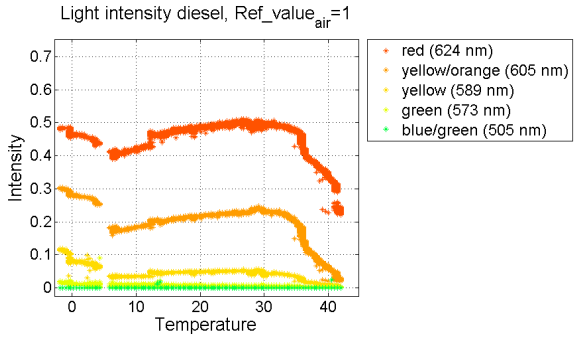


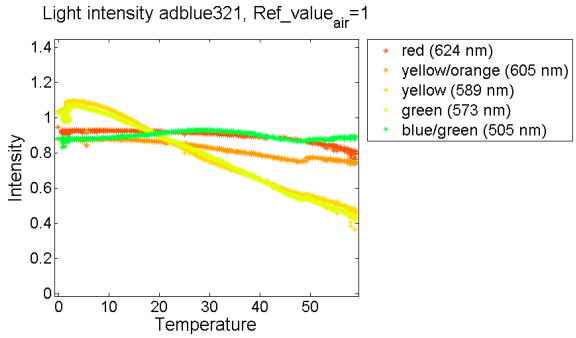
Figure 3.12: Top: Volt measured at Pin 4. Middle: Volt measured at Pin 3. Bottom: Volt measured at Pin 2.

3.2.7 Light absorption

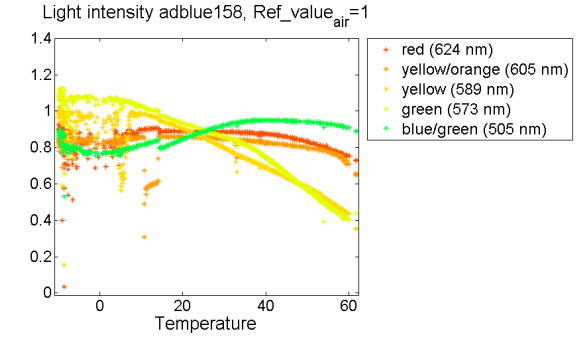
As for the conductivity measurement the measured light absorption values needed no prepossessing. However, the sensor had trouble with bubbles and coating forming on the PET tubes distorting the light. This resulted in the measured values not reflecting the actual light absorbing properties of the fluids. Using light absorption is regarded as interesting, but with the results obtained with the current prototype it was discarded as a feature for the classification system. Examples of measurements obtained using this sensor can be seen in Figure 3.13.



(a) Light intensity measured in diesel



(b) Light intensity measured in Adblue 32.1%



(c) Light intensity measured in Adblue 15.8%

Figure 3.13: Measurements obtained using the light absorption sensor

3.2.8 Temperature

The temperature in a fluid does of course not reveal any information about what kind of fluid it is in its own right, but it is important because other properties used as features change depending on the temperature. The measured voltage level was translated into a temperature level on the Celsius scale by calculating the corresponding resistance of the thermistor and using a lookup table provided by Wema relating the resistance to temperature.

3.3 Software for Feature Extraction

The Q-sensor-PCB alternates between sending four trigger pulses at 1 MHz and sending one shorter pulse. This shorter pulse is sent to check that the sensor is working properly. The LabVIEW program recording sensor data will also record data when this check pulse is sent. These recordings are removed in Matlab before feature extraction is performed. This is done by checking the trigger signal which is stored in the file *filename_us_trig.txt*. If only one pulse exists in this trigger signal the corresponding lines in the *filename_us_signal.txt* and *filename_arduino.txt* files are removed. The program is called *CleanUpRawData.m* and can be found by following the path specified in Appendix B.

The amount of data, collected in the lab, far exceeded the amount practical to work with when developing the feature extraction and classification code. Consequently, software was made to extract a certain number of samples from the complete set of raw data. The wanted number of data samples is specified by the user and the program will extract this number of samples evenly spread from the original set of raw data and save them in a new file. The program is called *Small_selection_of_files.m* and can be found by following the path specified in Appendix B.

To extract all the features previously presented a program was made

in Matlab reading all the raw data files from LabVIEW, extracting the features and saving them to text files to be further used by the classifier algorithms. In addition to the files containing the feature vectors, files containing the feature names were saved. The program is called *feature_extraction.m* and can be found by following the path specified in Appendix B. Most of the features could be extracted more or less straight forward, but the RTT detection algorithm has been subject to more extensive work, searching to make it more robust to noise. This work has been described in a Chapter 4.

Chapter 4

Development of a new RTT detection algorithm

In the system that is currently used to detect the urea concentration of Adblue, the urea concentration is related to the speed of sound in the fluid. This property is directly related to the RTT of the transmitted ultrasound pulse. Under certain conditions in the tank it has been noted that the current RTT detection algorithm detects sudden jumps in the RTT. These RTT jumps also have an impact on the urea concentration measurement. As the RTT was assumed to be of great importance as a feature for the classification algorithms to be presented later in this report, a new RTT detection algorithm was developed to eliminate the sudden jumps. The considerations done about the jump phenomenon and the development process will be presented in this chapter. At the end of the chapter the results of this work are presented together with a discussion of what was achieved.

4.1 RTT detection algorithm used in the current system

In the current system the RTT is detected by finding the start of the first echo using some criteria concerning the amplitude and frequency of a pulse in order to be recognized as a true echo and not noise. When the start of the echo is found, the point where the first zero crossing occurs is defined as the position of the echo.

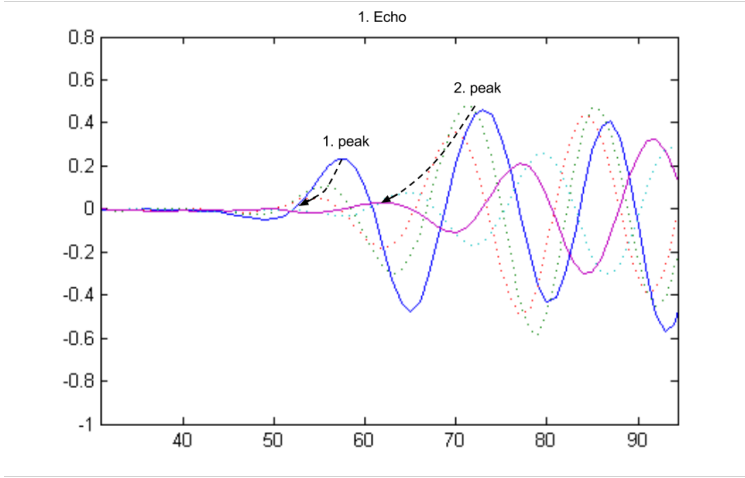


Figure 4.1: Distortion of echo by bubbles

The advantage of this method is the fact that it only uses the first echo of the signal in order to detect the RTT. This is advantageous because the presence of even small amount of bubbles in the fluid will have a strong impact on the echo strength and further make later echoes very hard, and sometimes impossible to detect. However, the presence of bubbles will not only weaken the echo, but also distort it, and make it shift in time. During practical use of the RTT detection it has been noticed that under certain conditions sudden jumps in the detected RTT occur. The size

of these jumps coincides with the time of one period in the transmitted signal. The reason for these sudden jumps is the signal being distorted in a way that is damping the first periods of the signal considerably more than later periods. This may lead to the first period being damped sufficiently to make it be discarded as noise, and hence result in a sudden jump of the detected RTT. An example of this phenomenon can be seen in Figure 4.1. The blue line is the start of an echo under good conditions without the presence of bubbles in the fluid. The purple line is the same echo after bubbles have been introduced in the fluid by using a pump. The dotted lines represent intermediate echoes. As we can see from the plot, the peaks shift towards the left. In the pink echo the peak of the first period has also been damped so much that it is barely visible. In this case what is actually the second period of the echo will be detected as the start of the echo by the zero crossing algorithm currently used in the system.

4.2 RTT detection algorithm comparing echoes

To prevent the sudden jumps from taking place in the RTT detection, an algorithm based on comparing the first echo with the second echo was tested. The idea behind this approach was that because both echoes pass through the same fluid with the same noise conditions they will both suffer from approximately the same distortion. Hence, if the first period of the first echo disappears so will the first period of the second echo.

A traditional way of comparing two echoes is by using correlation. One way is to find the autocorrelation [22] of the signal, written as:

$$R_{xx}(k) = E[x_{n+k}x_n] \tag{4.1}$$

In this equation x_n is sample number n in the ultrasound signal. This will return the highest correlation at $k = 0$, meaning that the signal correlates best with itself at zero time shift. Even complete random signals will

have this property. When k equals the RTT a new high peak in the autocorrelation will occur because the first echo aligns with the second echo, the second echo aligns with the third echo and so on. Alternatively one can roughly detect the first and second echo and calculate the cross correlation of two windows each containing one of the echoes.

$$R_{yx}(k) = E[y_{n+k}x_n] \tag{4.2}$$

Now x is the vector containing the first echo and y is the vector containing the second echo. x and y is extracted from the ultrasound signal with a certain time shift t. The k corresponding to the best correlation between the two vectors is then used to adjust the first rough estimate t. This will have the advantage of saving a number of computations, but it demands some preprocessing in order to extract the two echoes.

For algorithms based on correlation to work well and have clearly defined peaks at intervals corresponding to the RTT the transmitted signal should have a high correlation only when the echoes completely overlap. The signal being transmitted in today's sensor does not fulfill this property. An example of a signal with said property is a Chirp signal [15]. To demonstrate this a chirp signal and its autocorrelation was generated in MATLAB as can be seen in Figure 4.2. Algorithms using correlation have still proven to return accurate estimates as long as the conditions of the measurements are noiseless. If the measurements are done in an environment with a high noise level the correlation algorithms will lose their accuracy, however they still manage to return rough estimates missing with a certain number of periods. The reason why correlation algorithms easily miss the true RTT value by a period is because there is a very high correlation between an echo and the same echo shifted by the length of a period. As previously stated, if the signal being transmitted did not have this property, correlation algorithms would probably be more robust to noise. However, this would require a new transducer and will not be

discussed further in this report.

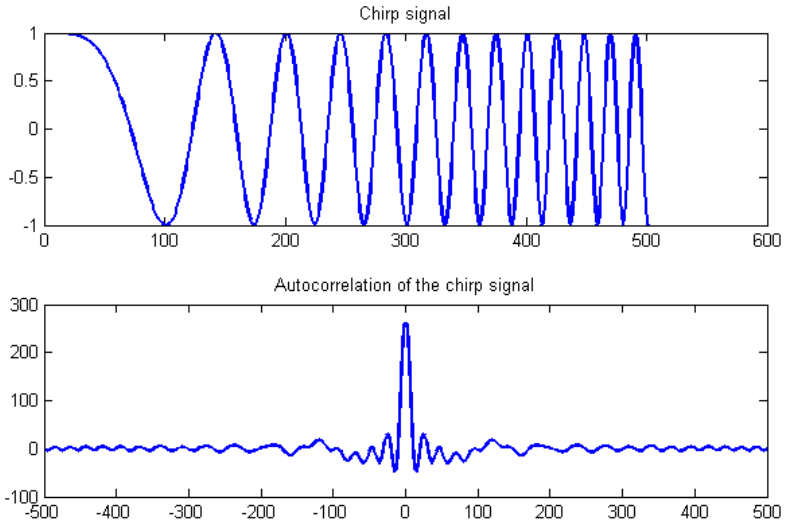


Figure 4.2: Chirp signal and its autocorrelation

Because the transmitted signal is not ideal for comparing echoes by using correlation, a different approach to find the best match between the first and second echo was developed. This method was used to detect RTT in the project of which this master thesis is a continuation [26]. This method has been given the name "Minimum Error". A quick explanation of how it works will be presented later on, as this algorithm has not been explained in the previous report. This is necessary because a new algorithm has been developed, as part of this master thesis, building on the "Minimum Error" algorithm.

4.3 Motivation behind avoiding errors in the detection of RTT

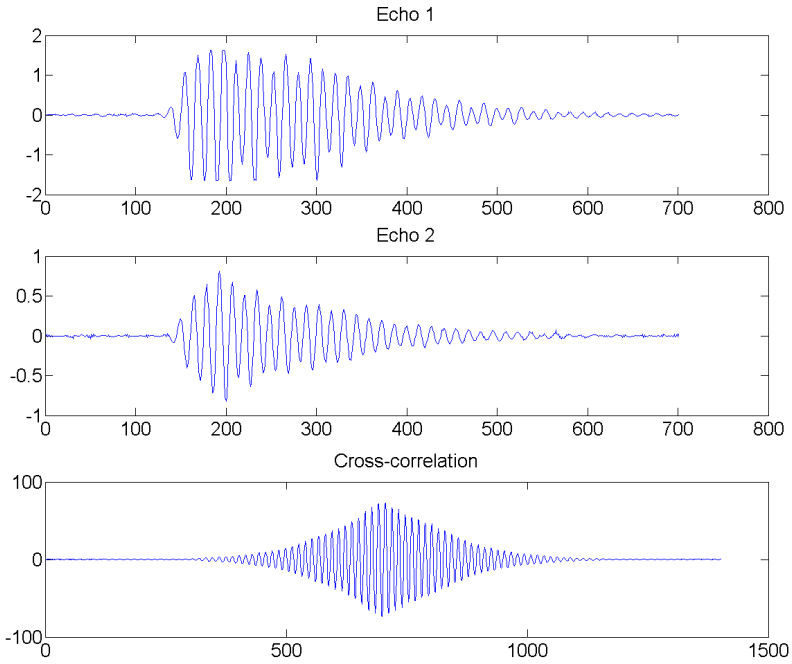


Figure 4.3: Correlation between first and second echo with today's transducer

As earlier mentioned, the motivation for developing a new algorithm to compare two echoes was that the transmitted signal from the currently used transducer is not ideal for using correlation. An example of this can be seen in Figure 4.3. Here the first and second echo of a signal has been detected and extracted. The cross-correlation of the two echoes has then been calculated and is shown in the bottom plot. Notice how there is no

clear peak in the correlation plot indicating the best mach. Instead there are many peaks more or less equally strong. This makes it difficult to know which peak is indicating the correct RTT.

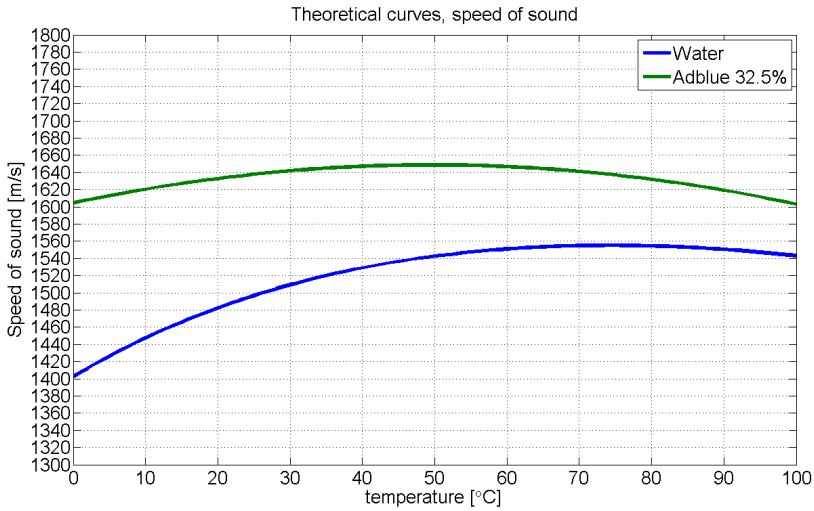


Figure 4.4: Speed of sound as function of temperature, Water and Adblue

Because the transmitted signal has a period of about 1 MHz, missing the overlap of the echoes by a period is equivalent to detecting the RTT with an error of 1 μs. The typical round trip time of the system lays around 70 μs. This results in an error of approximately 1.43%. This might seem as an insignificantly small detection error. However, the RTT itself is not of interest for the application of the system. It is used to calculate the urea concentration of the Adblue, which ideally should be 32.5%. The way this is done is by calculating the speed of sound based on the RTT and then compare this result to the theoretical speed of sound in water and Adblue. The speed of sound in fluids usually changes with the temperature in the fluid, thus the theoretical speeds of sound is a function of temperature as can be seen in Figure 4.4. The effect of missing by one period in the RTT

detection will not lead to a constant error in the concentration estimation. On the contrary it will be highly dependent of the true speed of sound in the fluid and of the temperature. The speed of sound in the fluid is related to the RTT by the formula:

$$v = \frac{d}{RTT} \tag{4.3}$$

Where v is the speed of sound and d is the travel distance of the sound. Now, to see how an error in the RTT detection affects the calculation of v, imagine the RTT is detected with an error of k seconds. The true RTT can be written as RTT_{true}, and the detected RTT can be written as:

$$RTT_{detected} = RTT_{true} + k \tag{4.4}$$

The true speed of sound is

$$v_{true} = \frac{d}{RTT_{true}}, \tag{4.5}$$

and the calculated speed of sound is

$$v_{detected} = \frac{d}{RTT_{detected}} = \frac{d}{RTT_{true} + k}, \tag{4.6}$$

The error of the calculated speed of sound can now be written as:

$$\begin{aligned} v_{error} &= v_{true} - v_{detected} \\ &= \frac{d}{RTT_{true}} - \frac{d}{RTT_{true} + k} \\ &= \frac{dk}{RTT_{true}^2 + RTT_{true}k} \end{aligned} \tag{4.7}$$

By finally inserting:

$$RTT_{true} = \frac{d}{v_{true}}, \tag{4.8}$$

the formula for the error in the calculated speed of sound v_{error} as a function of speed of sound v, travel distance d and RTT error k can be written as:

$$v_{error} = \frac{v^2 k}{d + vk}, \tag{4.9}$$

This shows that v_{error} approaches infinity when v goes to infinity. In other words the higher the speed of sound the bigger the error will be, given a fixed error in the RTT detection k. In addition, if the travel distance d gets bigger the impact of the RTT error will get smaller. This means that one way of minimizing the effect of error in the RTT detection is to increase the travel distance. Although this might seem as a good idea by simply looking at this formula, in practice it will lead to the sound traveling a longer distance in the fluid and hence enhance the impact of noisy conditions on the detected RTT. This might lead to a higher RTT error k, overall resulting in an increased v_{error}. In addition to the fact that an increased travel distance d will not necessarily result in a smaller v_{error}, increasing the distance will also result in the sensor being physically bigger. Whether or not this would be acceptable would have to be considered.

As the algorithms using correlation have a tendency to miss by multiple lengths of periods, this implies that $k = n/f,\ n = 0, \pm 1, \pm 2, \cdots$ where f is the frequency of the transmitted signal. By increasing the frequency of the transmitted signal each miss by a period, in the RTT detection will have a smaller impact on the calculated speed of sound. However, a change in frequency will also have an effect on attenuation of the sound in the fluid and the dispersion effect of bubbles. Thus, it is hard to conclude on the overall effects of increasing the transmitted frequency.

Changing the physical properties of the ultrasound sensor is beyond the scope of this report and the considerations presented here will not be investigated further. Instead the focus of this report will be on developing

software able to eliminate errors in the detected RTT caused by missing by a number of complete periods.

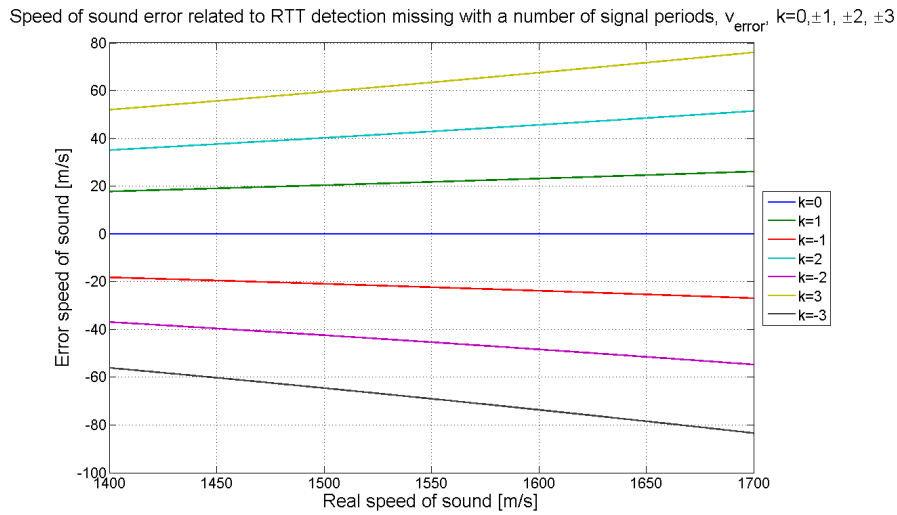


Figure 4.5: Error in the speed of sound detection given a miss in the RTT detection corresponding to a complete number of periods. Here the periods of the signal is of length 1 μs and the travel distance of the sound is 0.109 meters.

To demonstrate the effect of missing by a number of periods, Figure 4.5 shows v_{error} given $0, 1, 2$ and 3 period misses in the detected RTT given the true speed of sound being in the range 1400 to 1700 m/s. This range corresponds to the typical working range of the system, as can be seen by Figure 4.4, including the range of speed of sound in water and Adblue. The distance d used in the making of this plot is 0.109 meters, corresponding to the travel distance of today's sensor. As the speed of sound estimate is used to calculate the urea concentration in Adblue it is interesting to see how the detected speed of sound will be affected by these period misses. Using the theoretical values of speed of sound in Adblue, shown in Figure 4.4,

and adding the corresponding error due to period misses the plot shown in Figure 4.6 was made. This figure clearly demonstrates that missing by just a few periods will have a significant impact on the speed of sound estimates.

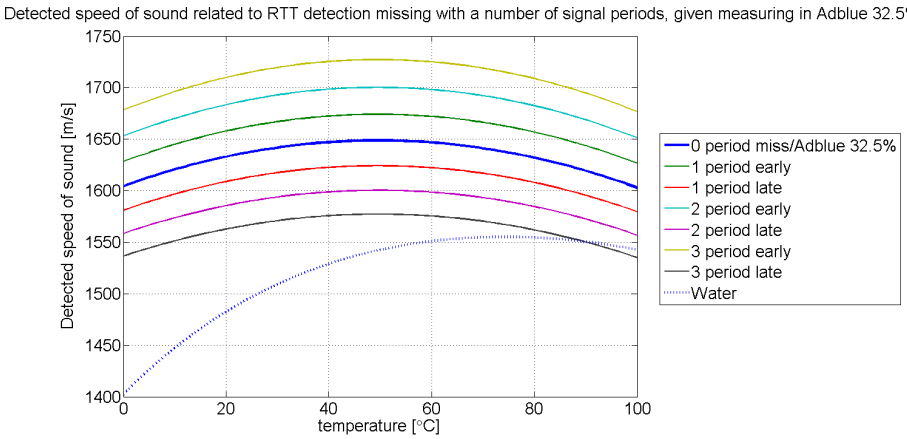


Figure 4.6: Speed of sound calculated when measuring in Adblue 32.5% and missing by a certain number of complete periods. The dotted line represents the theoretical speed of sound in water.

To calculate the urea concentration in the fluid the function presented in Equation 4.10 is used:

$$Urea\ concentration = 32.5\% \cdot \frac{v_{detected} - v_{water}}{v_{Adblue32.5\%} - v_{water}}, \tag{4.10}$$

In the case in which this function returns a negative value, zero will be returned instead, as a negative concentration of urea in water is not possible. In the cases in which this still happens it can either be due to the presence of other types of fluid in the tank with a speed of sound below the speed of sound in water, or it can be due to an error in the detected

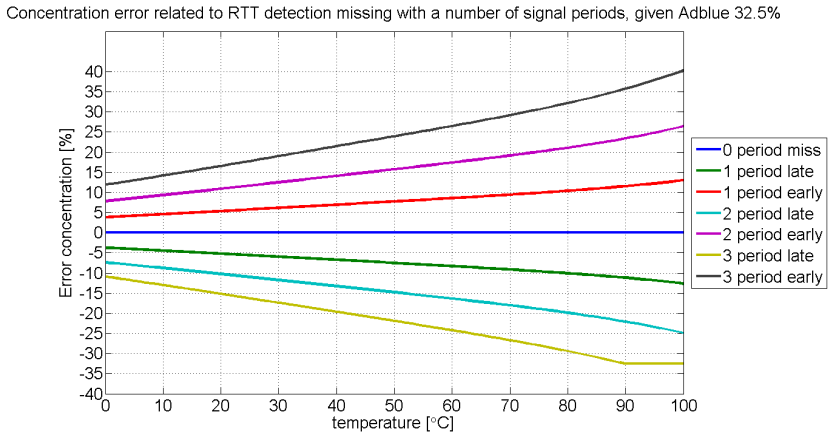


Figure 4.7: Error in reported urea concentration given a certain number of period misses in the RTT detection as function of temperature.

RTT. Assuming Adblue with a concentration of 32.5% urea is measured, the effect, on the reported concentration, of missing by a number of periods in the RTT detection is shown in Figure 4.7. As can be seen here, even missing by only 1 period will result in an error in the reported urea concentration of about $\pm 4\%$ to $\pm 13\%$, depending on the temperature in the fluid.

Based on these considerations it seems clear that avoiding such period misses in the detected RTT is crucial in order to report a reliable concentration measurement. In addition, it might also be crucial if the speed of sound in the fluid is to be used as a feature by a classification system detecting the presence of fluids not being Adblue. As a result of this, the development of a new algorithm for RTT detection will be presented in the following.

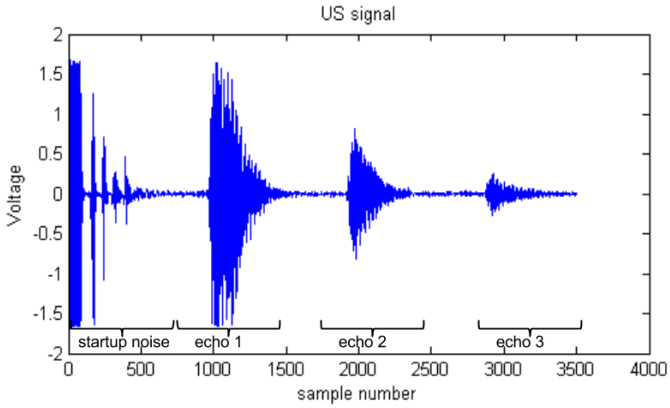


Figure 4.8: A typical example of the ultra sound signal

4.4 Minimum Error algorithm for detection of RTT

A typical signal can be seen in Figure 4.8. The idea behind the Minimum Error algorithm is to find the time shift minimizing the square of the difference between the first and the second echo of the ultrasound signal. This is done by initially extracting the first and second echo and storing them in two separate vectors, *echo1* and *echo2*. An error vector is then created using equation 4.11.

$$\begin{aligned} error(i) &= \sum_{n=1}^{N}(echo1(n) - echo2(n+i))^2, \\ N &= length(echo1), M = length(echo2) \\ i &= (1-N), (2-N), \cdots, (M-1) \end{aligned} \tag{4.11}$$

The index of the minimum value in the error vector indicates the shift between the two vectors resulting in the best overlap. This index is now used to correct the time shift of the initial extraction of *echo1* and *echo2*. A step by step walkthrough of the algorithm will now be presented:

1. Remove startup noise. See Figure 4.9

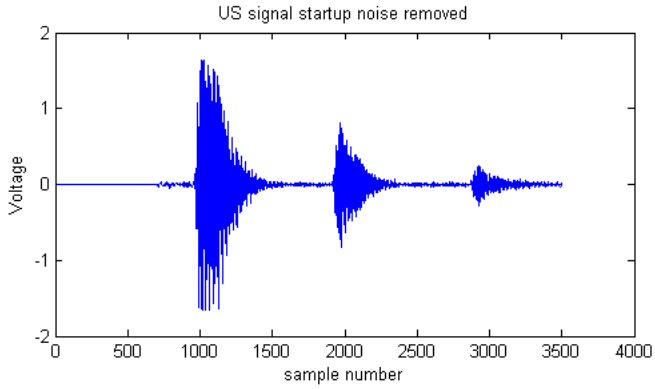


Figure 4.9: Ultrasound signal without the startup noise, due to internal ringing in the piezo.

2. Calculate the signal energy distribution of the signal, see Figure 4.10a and Equation 4.12.

$$energy(n) = \sum_{i=0}^{28} ultrasound\_vector(n+i)^2 \qquad (4.12)$$

3. Look for the highest energy peak in the area where echo 1 can be located, see Figure 4.10b.

4. Roughly determine the start of *echo1* (move back from max peak until a threshold is crossed), see Figure 4.10b. The threshold is a dynamic value so that 20% of the vector elements are stronger than the threshold.

5. Look for the highest energy peak where *echo2* should be located. This is adapted by the rough location of *echo1*. (Should be twice the distance), see Figure 4.10c.

6. Detect the start of *echo2* roughly (in the same way as for *echo1*),

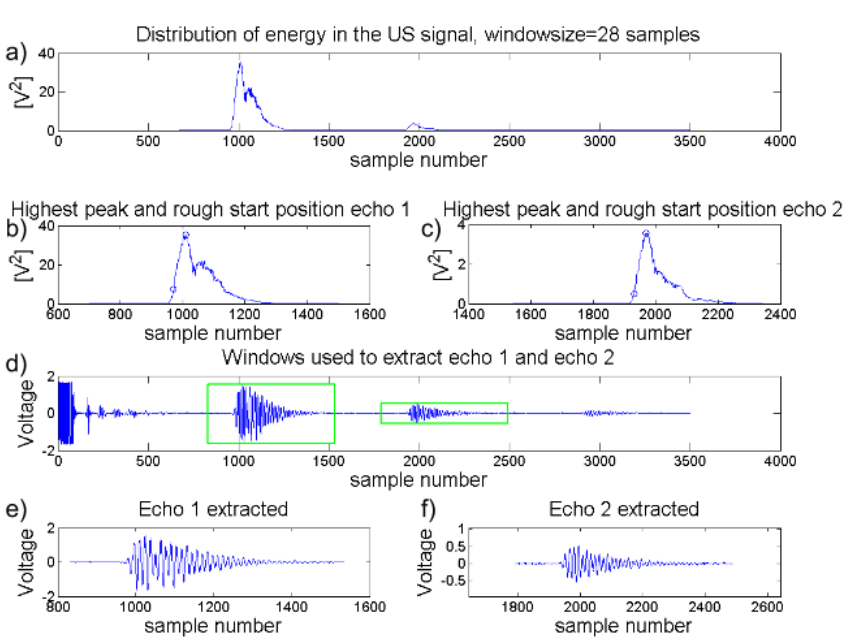


Figure 4.10: Steps to extract *echo1* and *echo2* from the ultrasound vector

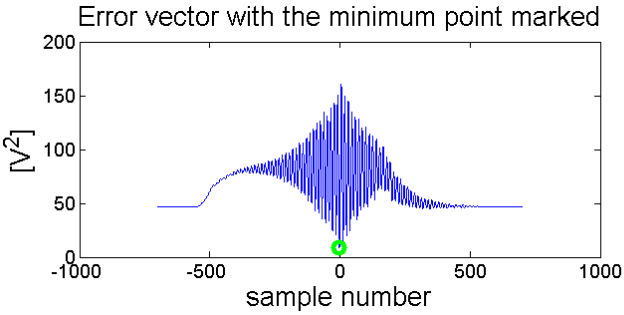
see Figure 4.10c.

7. Store *echo1* and *echo2* in two vectors using a window around the rough estimate of their location, see Figure 4.10d,e and f.

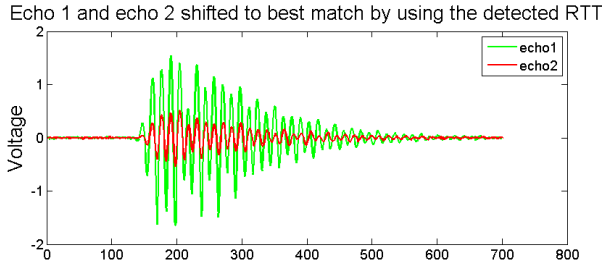
8. Generate the error vector using equation 4.11, see Figure 4.11a.

9. Use the index of the minimum error together with the start indexes of the two vectors containing *echo1* and *echo2* to calculate the RTT, see Figure 4.11b and Equation 4.13.

$$RTT = index + start\_echo2 - start\_echo1 \tag{4.13}$$



(a) Error vector



(b) Best match between *echo1* and *echo2*, using the minimum error algorithm

Figure 4.11: Minimum error algorithm to find best match between the first and second echo of a ultrasound signal

4.5 Results using the Minimum Error algorithm for detection of RTT compared to correlation and zero crossing

The results using the Minimum error algorithm turn out to be the same as using correlation as can be seen in Figure 4.12. A visual inspection of what happens in the algorithms shows that both algorithms works as intended but have the same difficulties caused by the distortion of the second echo being stronger than of the first echo. This leads to the sudden jumps in the detected speed of sound around $40°C$. It is important to emphasize

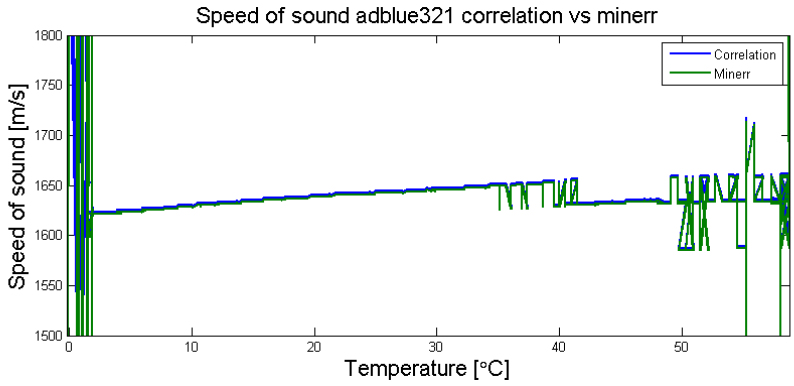


Figure 4.12: Speed of sound detection, cross-correlation vs. minimum error

that the jumps are not due to the temperature in the fluid itself but due to the conditions in the tank at this moment.

Because the original algorithm based on zero crossing detection only uses the first echo, it handles noise slightly better as can be seen in Figure 4.13. But the zero crossing algorithm also fails, and when it does the error is slightly bigger.

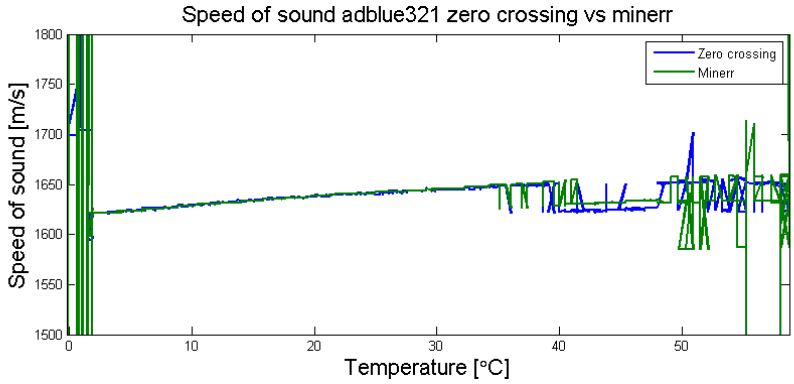


Figure 4.13: Speed of sound detection, zero crossing vs. minimum error

4.6 Improving the RTT detection algorithm

Although the sudden jumps in the detected RTT reflect the correct behavior of all the algorithms so far treated in this report, sudden jumps in the speed of sound do not occur. What happens is a distortion of the signal gradually occurring over some time. The reason why all the algorithms detect jumps instead of a smooth transition is that they all use the periods in the signal to define the presence of an echo, and there will always be some fixed limit for when the amplitude of a period is strong enough to be perceived as part of an echo.

If an algorithm could be made comparing the position of the current signal to the position of the last signal measured, one could in theory calculate the rate of change in the position of the signal and sum this up over time to keep track of the speed of sound in the fluid.

$$RTT_i = RTT_{i-1} + dRTT_i \tag{4.14}$$

In Equation 4.14 RTT_i is the current RTT, RTT_{i-1} is the last RTT detected and $dRTT_i$ is the change detected between the last and the current measurement. To find $dRTT_i$ one can use cross-correlation (Equation 4.2) between the last and the current measurement of the signal to find the lag, k, resulting in the best correlation. Or alternatively the minimum error approach can be used (Equation 4.11), but then finding the index i resulting in the minimum error. The detected change of RTT can then be calculated as:

$$dRTT_i = \frac{k}{Fs} \tag{4.15}$$

Fs being the sampling frequency. The advantage of this method is that a complete signal can be compared to a complete signal or alternatively one can simply compare two first echoes. Because the distortion of the signal

occurs over some time, two following signals will result in a good correlation. However, there are two obvious problems with this approach. One being how to define the initial RTT value. The second being drifting caused by rounding errors in the numeric calculations. With a sampling frequency, $Fs = 14.29MHz$, the minimum RTT step $dRTT$ would be approximately $0.07\mu s$ with an error, $dRTT_{error}$, in the range $dRTT_{error} \leq \pm 0.035\mu s$.

In an attempt to combine the advantages from the different methods and circumvent their disadvantages, an algorithm combining different approaches is suggested. The algorithm will use parts off the Minimum error algorithm previously explained, finding the time shift between the first and second echo. In addition, a reference signal will be stored in order to compare the complete signal to this reference signal and hence be able to detect the change of the RTT over time. The reference signal will be updated occasionally when certain conditions of the signal are met. The new algorithm will in the following be referred to as the *Improved Round-trip-time Detection*, IRD.
The IRD works as follows:

- Calculate the signal variance, σ^2.
- Update the signal variance threshold, σ_t^2, according to the following equation:

$$\sigma_t^2 = max\{\sigma_t^2, 0.99\sigma^2\} \tag{4.16}$$

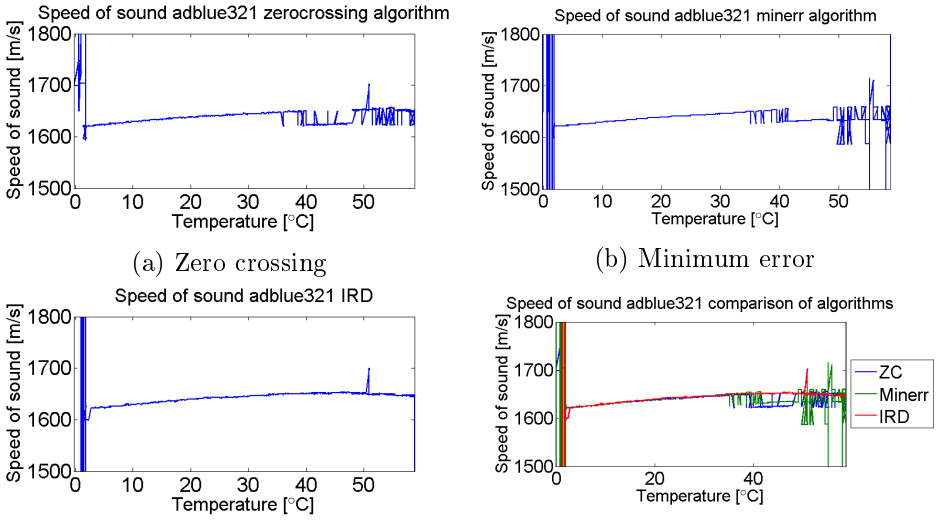
- If $\sigma^2 > \sigma_t^2$ calculate the RTT as described in section 4.4. Store the current signal as a reference signal together with the detected RTT.
- If $\sigma^2 \leq \sigma_t^2$ find the shift, $dRTT_i$, between the current signal and the reference signal. Calculate the new RTT measurement using Equation 4.14. If $0.84\mu s \leq abs(RTT_i - RTT_{i-1}) \leq 1.12\mu s$ update the reference signal to the previous signal together with RTT_{i-1} and

recalculate RTT_i using the new reference value. This is done because jumps of about $1\mu s$ are probably due to distortion of the signal.

4.7 Results using the IRD algorithm for detection of RTT compared to Minimum Error and zero crossing

The results using the IRD algorithm have through testing shown to be more robust to avoid sudden jumps in the detected speed of sound. The combination of RTT detection using both echoes when conditions are believed to be good, and using a reference signal when conditions are believed to be harsh have proven to yield promising results. A demonstration can be seen in Figure 4.14 where three algorithms have been applied to the same piece of raw data. Both the zero crossing and the minimum error algorithm suffer from sudden jumps in the detected speed of sound whilst the IRD algorithm returns a smooth curve with the exception of one outlier at about $50°C$. At low temperatures near zero degrees none of the algorithms returns valid results, this is due to parts of the ultrasound sensor being located in a block of ice, and no valid echoes are returned to the sensor.

To further evaluate the performance of the IRD algorithm compared to the zero crossing algorithm both algorithms were tested with ultrasound signals retrieved by measuring over a broad temperature band in all the fluids in question. The test was done by comparing the RTT returned by the algorithms to the correct RTT value. In order to make such a comparison the correct RTT value needed to be found. The speed of sound in a fluid changes with the temperature. As a consequence of this, the speed of sound can be seen as function depending on temperature. One way to define the correct values is to use polynomial regression to



(a) Zero crossing

(b) Minimum error

(c) IRD

(d) Comparison

Figure 4.14: Comparison of speed of sound detection using Zero Crossing, Minimum Error and the IDR algorithm

define polynomial functions best fitting the recorded data. Coefficients for a fifth order polynomial function have already been found for water, hereby referred to as the Marczak-coefficients [20]. For the other classes second order polynomials were found using data recorded at the Wema lab in Bergen. This was done in the following way:

- For all the fluids considered the ultrasound signal was recorded over a wide temperature range using today's ultrasound sensor. The temperature was recorded by using the thermistor integrated in the ultrasound sensor.
- The RTT was detected by using both the zero crossing algorithm and the IRD algorithm. All the data points were plotted as a function of temperature, and the obvious outliers were removed manually before using the MATLAB function *polyfit* [24] to find the coefficients best

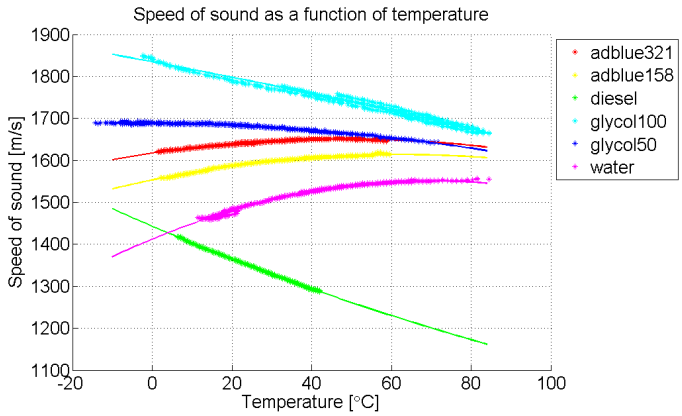


Figure 4.15: Speed of sound curves approximated as second order polynomials for the fluid classes as function of temperature

Order:	2th	1th	0th
Adblue321	−0.0146	1.4007	1617.0
Adblue158	−0.0158	1.9624	1553.1
Diesel	0.0084	−4.0526	1443.0
Glycol100	−0.0026	−1.7873	1834.5
Glycol50	−0.0079	−0.1207	1689.1

Table 4.1: Coefficients for speed of sound curves based on recorded data

fitting the data.

The resulting curves can be seen in Figure 4.15. Here * indicates a data point used to find the polynomials. The corresponding coefficients can be seen in Table 4.1. By using these curves as reference the two RTT detection algorithms were tested. Both algorithms have a tendency to miss by a number of signal periods. This corresponds to about 14 samples in the raw ultrasound signal. The threshold for counting a RTT detection as erroneous was therefore chosen to be missing by more than 7

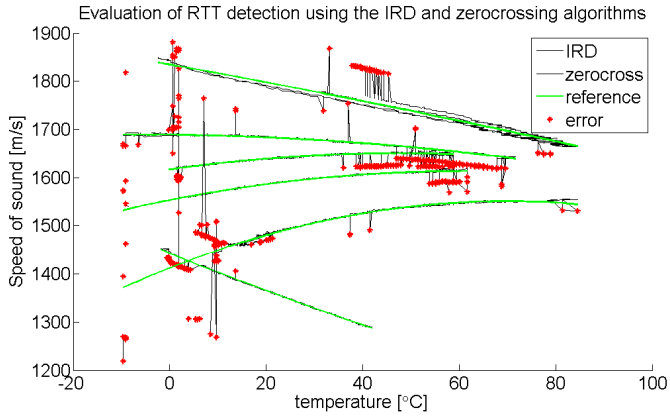


Figure 4.16: Evaluation of RTT detection using measurements from different types of fluids

samples compared to the reference value. The data set used to evaluate the two algorithms can be seen in Figure 4.16. In Table 4.2 the performance of the two algorithms has been presented. Here it can be seen how the two algorithms performed on a data set consisting of 5537 measurements at different temperatures and in different fluids. The evaluation of the algorithms has three possible outcomes:

Algorithm	Total	Invalid	Valid	Correct	Error
IRD #:	5537	778	4759	4343	416
IRD %		14.1	85.9	91.3	8.7
Zerocross(ZC) #	5537	877	4660	4213	447
Zerocross(ZC) %		15.8	84.2	90.4	9.6
IRD and/or ZC #	5537	419	5118	5004	114
IRD and/or ZC %		7.6	92.4	97.8	2.2

Table 4.2: Evaluation of the IRD and Zerocrossing algorithms

- *Invalid.* The algorithm fails to detect the RTT caused by too much noise or no signal.
- *Correct.* The algorithm returns an RTT value less than 7 samples off the reference value.
- *Error.* The algorithm returns an RTT value more than 7 samples off the reference value.

In addition to presenting the performance of the two algorithms a combined performance can be seen. In this case the three outcomes corresponds to the following:

- *Invalid.* Both algorithms fail to detect the RTT caused by too much noise or no signal.
- *Correct.* One or both of the algorithms returns an RTT value less than 7 samples off the reference value.
- *Error.* Both algorithms return an RTT value more than 7 samples off the reference value.

The table shows that the IRD algorithm performs slightly better than the Zero crossing algorithm. It returns more valid results and when it does its success rate is higher than for the Zero crossing algorithm. However, their performances are quite close. An interesting observation is that their combined performance is significantly better than their separate performance. This indicates that the two algorithms perform well under different kinds of conditions. The number of invalid results are about half of what either algorithm manages separately and 97.8% of the valid results are correct. Using both algorithms in parallel could be an option to obtain a more robust RTT detection. In the case where only one of the algorithms returns a valid result, this result is used. If both algorithms return valid but different results a decision would have to be made as to which result should

be used. However, the results shown in Table 4.2 demonstrate an ideal situation where the algorithms are combined in a way so that the correct result is always selected if present. Finding a good way to combine the two algorithms would be of interest, but further improvement of the RTT detection algorithm is not pursued in this report. Based on these results the IRD algorithm was used to detect speed of sound as a feature for the classification system.

Chapter 5

Development of Classifier Software

5.1 Minimum-error-rate

The minimum error rate classification algorithm is based on assuming a probability distribution and define classification rules to minimize the probability of erroneous classifications. The parameters of the assumed probability distribution are estimated using a training set consisting of feature vectors with known classes.

5.1.1 Feature space based on physical features

Previously in this report a great number of features has been presented, but it has not been investigated if they are all linearly independent. If two or more features turn out to be linearly dependent this implies that they provide the same piece of information about the problem, and hence the number of features can be reduced without a loss of information. Reducing the number of features implies a reduction of the dimensionality of the problem. This is positive as it reduces computational cost. In addition to

the advantage of working with less features it is also necessary to remove linearly dependencies from the set of features when using the Minimum-error-rate classifier. This is because the covariance matrix that will be used to define the decision rule can not be singular. The reason for this can be seen in Equations 2.4, 2.5 and 2.6. In these equation the covariance matrix is inverted. A singular matrix is not invertible. If two linearly dependent variables are both kept when calculating the covariance matrix, this will lead to the covariance matrix being singular. In addition to the necessity of removing linear dependencies from the covariance matrix to avoid it being singular the features should also be scaled properly. If the features are badly scaled the covariance matrix might be close to singular, which in turn will lead to noise in the feature vectors being amplified and obscure the results. As the features used in this classification problem vary in scalar size and units it is clearly necessary to preform some kind of scaling in order to combine the features in a meaningful way. Unfortunately, there is no standard indisputable way to scale the features in order to avoid a badly scaled covariance matrix [12]. As a result, the scaling was performed based on knowledge about the scale and natural range of the different features. The goal of the scaling is to obtain features with numerical entries with a reasonable relative scaling. To keep the original numeric values as intact as possible the scaling was limited to factors of ten. The temperature scale was used as the default scaling. All other features were scaled to approximate this scale.

Having scaled the features, and thus hopefully having avoided problems related to a near singular covariance matrix caused by badly scaled features, linear dependencies between features were explored. This was done by generating a covariance matrix based on the set of features for each class. The rank of each covariance matrix was then calculated to find out how many of the features are independent. However, the covariance matrix is only meaningful when working with normally distributed data.

	min:	max:
Temperature	-13.581	84.54
Conductivity Pin 4	0	3.087
Conductivity Pin 3	0	3.41
Conductivity Pin 2	0	5.013
Startup noise peak strength Peak 1	0.19968	1.6685
Startup noise peak strength Peak 2	0.14323	1.6726
Startup noise peak strength Peak 3	0.07252	1.637
Startup noise peak strength Peak 4	0.050606	1.6371
RTT	55.983	85.934
Max amplitude echo1	0.205	1.6845
Max amplitude echo2	0.025	1.2715
Damping factor between echo1 and echo2	0.91	8.74
Max peak frequency spectrum	1.3935e-08	4.2205e-06
Factor between 1 MHz and 600KHz echo1	1.6561	24.028
Factor between 1 MHz and 600KHz echo2	0.52369	20.338
Signal variance	0.001018	0.14933
Signal variance after startup noise	0.001018	0.14933

Table 5.1: Minimum and maximum values for the features with original scaling

Using the original scaling the minimum and maximum values, when including all classes, of the different features can be seen in Table 5.1. Based on these values a scaling vector was manually chosen, it is shown in Equation 5.1.

$$scaling\_vector = [1\,10\,10\,10\,10\,10\,10\,10\,1\,10\,10\,10\,1e7\,1\,1\,100\,100] \quad (5.1)$$

This vector was then used to scale all the vectors in the set of feature vectors. The resulting minimum and maximum values of the different features can be seen in Table 5.2.

	min:	max:
Temperature	-13.581	84.54
Conductivity Pin 4	0	30.87
Conductivity Pin 3	0	34.1
Conductivity Pin 2	0	50.13
Startup noise peak strength Peak 1	1.9968	16.685
Startup noise peak strength Peak 2	1.4323	16.726
Startup noise peak strength Peak 3	0.7252	16.37
Startup noise peak strength Peak 4	0.50606	16.371
RTT	55.983	85.934
Max amplitude echo1	2.05	16.845
Max amplitude echo2	0.25	12.715
Damping factor between echo1 and echo2	9.1	87.4
Max peak frequency spectrum	0.13935	42.205
Factor between 1 MHz and 600KHz echo1	1.6561	24.028
Factor between 1 MHz and 600KHz echo2	0.52369	20.338
Signal variance	0.1018	14.933
Signal variance after startup noise	0.1018	14.933

Table 5.2: Minimum and maximum values for the features after scaling with the scaling vector seen in Equation 5.1

Clearly, the temperature in the fluid is not normally distributed as it will vary in a wide range affected by the surrounding environment. The surrounding environment will in an operating system consist of the vehicle at which the tank is mounted. The temperature in the fluid will, therefore, change according to many factors, including the time of year, how long the engine has been running, and the current workload of the engine. The temperature is, therefore, not really a feature of the fluid being measured, but a factor influencing some of the other features. Consequently, it must be accounted for in spite of not being considered a feature. It is well known that temperature affects the speed at which sound propagates through fluids [20]. Although in theory a fluid will have a true speed of sound at a given temperature, in a practical situation, where the speed of sound is measured, there will always be uncertainties related to the measured speed of sound. The RTT feature is directly related to the speed of sound by Equation 4.3. As a consequence of this the RTT feature can be modeled as a normally distributed stochastic variable with an expected value equal to the assumed true RTT. Before calculating the covariance matrix the RTT feature was therefore normalized by subtracting the expected RTT value based on the speed of sound curves earlier presented in this report (Figure 4.15), for water the curve defined by the Marczak-coefficients were used [20]. The expected RTT value for each class was then defined as a function of temperature. Concerning the other features little is known about their relation to temperature. Although some of the features seem to have a certain correlation to temperature it is believed that other factors as bubbles in the fluids have a greater impact on these properties. They are, therefore, in the following assumed to be independent of temperature. As the temperature was only used to adjust the RTT feature it has not been included as a feature when calculating the covariance matrix. Consequently six covariance matrices were estimated, one for each class, using a feature set created by the program *feature_ extraction.m* (explained in

Section 3.3). For all the covariance matrices the rank was found to be 16 which is the same as their dimensionality. In other words the matrices all had full rank. This implies that the features are all linearly independent. However, there might still be linear dependencies hidden by noise in the retrieved sensor data. An indication of this could be high covariance between two features. In addition, if a covariance matrix has some very small eigenvalues, this can be an indication that such linear dependencies exist, but have been hidden by noise. If such dependencies exist between features it will most likely be discovered by systematic testing of the classifiers performance using different combinations of features. If adding or removing a certain feature has little or no effect on the overall performance of a classifier this means that the feature either contains no relevant information about the classes or it provides information already contained by an other feature.

Finally, a classifier based on Minimum-error-rate assuming normally distributed data was trained. This was done by estimating the expected value, $\boldsymbol{\mu}_i$, and covariance matrix, $\boldsymbol{\Sigma}_i$, for all the classes, i being the class index. For all i $\boldsymbol{\mu}_i$ is a column vector of size 16 and $\boldsymbol{\Sigma}_i$ is a matrix of size 16×16, 16 being the number of features. As the expected RTT value depends on temperature, $\boldsymbol{\mu}_i$ can be written as a function of temperature as demonstrated in Equation 5.2.

$$\boldsymbol{\mu}_i(t) = \begin{bmatrix} \mu_{i,1} \\ \vdots \\ \mu_{i,7} \\ \mu_{i,8}(t) \\ \mu_{i,9} \\ \vdots \\ \mu_{i,16} \end{bmatrix} \tag{5.2}$$

Class index:	class:	Expected RTT [μs] function:
1	Adblue321	$\mu_{1,8}(t) = \frac{10.9 \cdot 10^4}{-0.0146t^2 + 1.4007t + 1617.0}$
2	Adblue158	$\mu_{2,8}(t) = \frac{10.9 \cdot 10^4}{-0.0158t^2 + 1.9624t + 1553.1}$
3	Diesel	$\mu_{3,8}(t) = \frac{10.9 \cdot 10^4}{0.0084t^2 - 4.0526t + 1443.0}$
4	Glycol100	$\mu_{4,8}(t) = \frac{10.9 \cdot 10^4}{-0.0026t^2 - 1.7873t + 1834.5}$
5	Glycol50	$\mu_{5,8}(t) = \frac{10.9 \cdot 10^4}{-0.0079t^2 - 0.1207t + 1689.1}$
6	Water	$\mu_{6,8}(t) = \frac{10.9 \cdot 10^4}{a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0}$*

*a_0, a_1, a_2, a_3, a_4 *and* a_5 *are the Marczak-coefficients[20]*

Table 5.3: Functions defining the expected RTT values as a function of temperature

$\mu_{i,1}, \ldots, \mu_{i,7}, \mu_{i,9}, \ldots, \mu_{i,16}$ are constants for all i. $\mu_{i,8}(t)$ are defined by the functions shown in Table 5.3. The resulting probability density functions $p(RTT|temperature, class)$ is shown together in a 3d plot in Figure 5.1. Note that $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$, $i = 1, \ldots, 6$, really defines six 16-dimensional normal distributions varying with temperature. However, visualizing this in a plot would not make sense.

5.1.2 Evaluation of Minimum-error-rate classifier

With 16 available features, this allows for 65535 unique combinations of features when an arbitrary number of features are to be used. This can be derived by summing the binomial coefficients [25] $\binom{n}{k}$ for $n = 16$ and $k = 1, .., 16$ as can be seen in Equation 5.3.

$$\sum_{k=1}^{n} \frac{n!}{(n-k)!k!} \tag{5.3}$$

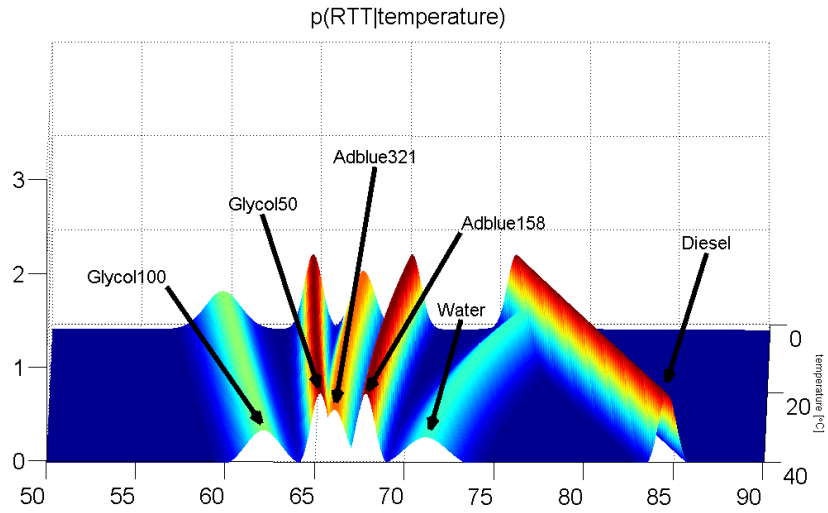


Figure 5.1: $p(RTT|temperature)$ for all classes of fluid

For all these 65535 combinations the classifier was trained and tested. The training of the classifier was done by the code seen in Program Code 1. Firstly, in order to avoid over fitting the classifier to the training data, a minimum standard deviation value *min_sigma* was chosen. Secondly the vector $\boldsymbol{\mu}_i$ was defined as a function of temperature. This is done in line 6 to 26 in the code. Further on, the covariance matrix is estimated and all the diagonal elements are adjusted to fulfill the minimum standard deviation property. Lastly the discriminant function parameters are calculated according to Equation 2.4, 2.5 and 2.6. These parameters are saved to a new cell structure *X_class_parameters*. A discriminant function taking in a feature vector and the classification parameters along with the temperature was defined as shown in Program Code 2. Using this function and a test set, the the classifier was tested by the code seen in Program Code 3. Here c is a confusion matrix [27]. The confusion matrix has the dimension $c \times c$, c being the number of classes. Each column corresponds to

```
%Training of the classifier====================================
min_sigma=0.5;
%mu equals the mean for all features except for the RTT as this
%is a function of temperature.
for i=1:size(X_class,1),
  expected_RTT=@(t,class_index)...
  1e6*(0.109./SOUND_VELOCITY_CUSTOM_POLYNOM(...
  poly_coeff(class_index,:), t));
  if length(features_used)==1 & features_used==10,
    mu=@(t) [expected_RTT(t,i)];
  else
    mu=mean(X_class{i});
    function_index=find(features_used==10);
    if isempty(function_index),
      mu=@(t) [mu(features_used)'];
    elseif function_index==1,
      mu=@(t) [expected_RTT(t,i); ...
      mu(features_used(function_index+1:end))'];
    elseif function_index==length(features_used),
      mu=@(t) [mu(features_used(1:function_index-1))'; ...
      expected_RTT(t,i)];
    else
      mu=@(t) [mu(features_used(1:function_index-1))'; ...
      expected_RTT(t,i); mu(features_used(function_index+1:end))'];
    end
  end
  coMat=cov(X_class{i}(:,features_used));
  for j=1:size(coMat,1),
    coMat(j,j)=max([coMat(j,j),min_sigma^2]);
  end
  %Define the discriminant function parameters=================
  w0=@(t) -0.5*(mu(t)'/coMat)*mu(t) ...
  -0.5*log(det(coMat))+log(Pclas);
  w_=@(t) coMat\mu(t);
  W=-0.5*inv(coMat);
  X_class_parameters{i}={W,w_, w0,mu};
end
%================================================================
```

Program Code 1: Training of Minimum error rate classifier.

```
function [ g_max, i_max ] =...
g_minimum_error_rate( feature, X_class_parameters, temperature )
%g_minimum_error_rate Returns the classindex of the most likely class.
%[ g_max, i_max ]
%g_max: value of the g_function with highest value.
%i_max: class index
  number_of_classes=size(X_class_parameters,1);
  number_of_features=size(X_class_parameters{1}{1},1);
  W=zeros(number_of_features,number_of_features,number_of_classes);
  w_=zeros(number_of_features,number_of_classes);
  w0=zeros(number_of_features,number_of_classes);
  %Save parameters for the discriminant function:====
  for i=1:number_of_classes,
    W(:,:,i)= X_class_parameters{i}{1};
    w_(:,i)=X_class_parameters{i}{2}(temperature);
    w0(i)=X_class_parameters{i}{3}(temperature);
  end
  %=================================================
  g_max=-Inf;
  i_max=0;
  for i=1:number_of_classes,
    temp=feature'*W(:,:,i)*feature+w_(:,i)'*feature+w0(i);
    if temp>=g_max,
      g_max=temp;
      i_max=i;
    end
  end
end
```

Program Code 2: Discriminant function for Minimum error rate classifier.

```
%initialize confusion matrix===================================
c=zeros(size(X_class_parameters,1));
%perform test of classifier
for i=1:size(X_test,1),
  [classification_value, classification_index]...
  =g_minimum_error_rate( X_test(i,features_used)'...
  , X_class_parameters, X_test(i,2));
  c(X_test(i,1),classification_index)...
  =c(X_test(i,1),classification_index)+1;
end
%Calculate error rate==========================================
P_error=sum(sum(c.*(ones(size(X_class_parameters,1))...
-eye(size(X_class_parameters,1)))))/sum(sum(c));
%=============================================================
```

Program Code 3: Code for testing of the Minimum error rate classifier.

the assigned class index of an object in the test set and each row corresponds to the true class index of the object. For each object in the test set the position defined by the column of the assigned class index and the row of the true class index is incremented. A correctly classified instance from the test set will, therefore, lie on the diagonal of the matrix, while a wrongly classified instance will lie outside of the diagonal. The advantage of presenting the result in this way is that it will clearly show if two classes are being mixed up by the classifier. From the confusion matrix P_{error} could be calculated by dividing the sum of all elements in the matrix not on the diagonal by the sum of all the elements in the matrix, see Equation 5.4, 5.5 and 5.6. For each number of features used, the combination of features resulting in the lowest numbers of erroneous classifications were stored. In addition to this, the corresponding confusion matrix (denoted C in the following) and the estimated chance, P_{error}, of making an erroneous classifications was stored. The resulting output can be seen in Appendix A.1.1.

$$Error = \sum_{i=1}^{c} \sum_{j=1}^{c} C_{ij}, i \neq j \tag{5.4}$$

$$All = \sum_{i=1}^{c} \sum_{j=1}^{c} C_{ij} \tag{5.5}$$

$$P_{error} = \frac{Error}{All} \tag{5.6}$$

As the Minimum-error-rate classifier is a parametric method assuming a certain probability distribution it was initially trained and tested using the same data. This was done to see if the assumed probability distribution managed to describe the actual distribution of the data well enough to make a decent classification of the data. If the classifier does not manage to correctly classify the same data that was used to train it, then it will not

help to test it using an independent data set. Furthermore, it will reveal if poor assumptions was made to train the classifier. As can be seen from the output of this evaluation strategy (Appendix A.1.1), the lowest error rate, P_{error} was achieved when only using the feature RTT to classify the data. This resulted in an error rate of 6.76%. Adding more features to the classifier resulted in higher error rates, as can be seen in Figure 5.2. This shows that adding the extra features actually contribute as noise to the classifier. This might be because it was assumed that only the RTT feature varied with the temperature. The other features might still contain relevant information about the classes of fluid, but this information might have been lost due to this oversimplification.

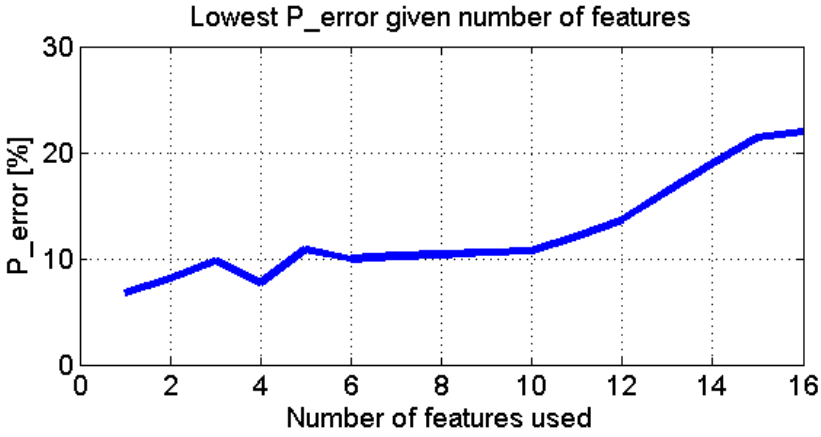


Figure 5.2: Minimum error rates P_{error} achieved by the Minimum-error-rate classifier assuming normally distributed features. 1 to 16 features used.

5.2 Reducing dimensionality of feature space

In an attempt to reduce the number of features while still maintaining most of the information in the original feature set, principal component analysis [16] was considered. However, because principal component analysis does not distinguish between the classes when finding the principal components the method might not find the best components to make a good classification. This will be the case when two classes have some significant features in common and a less significant feature separating them. To avoid this problem Multiple Discriminant Analysis [13] was tested.

5.2.1 Multiple Discriminant Analysis (*MDA*)

MDA has the advantage that it maximizes the between-class scattering while it at the same time minimizes the within-class scattering. The details of the method will not be discussed here but some key equations will be presented as they have been used in the software written to test this procedure. The goal of the method is to find a matrix, $\boldsymbol{W}$, with dimensions $d \times (c-1)$, where d is the original number of features available, and c is the number of classes to be recognized. The columns in, $\boldsymbol{W}$, will be referred to as $\boldsymbol{w}_i$. $\boldsymbol{W}$ will be used to transform the set of feature vectors, $\boldsymbol{x}$, to a new set of transformed feature vectors, $\boldsymbol{y}$, see Equation 5.7.

$$\boldsymbol{y} = \boldsymbol{W}^t \boldsymbol{x} \tag{5.7}$$

Two matrices are defined, used as a measure of the within-class scatter, $\boldsymbol{S}_W$ (see Equation 5.8), and the between-class scatter,$\boldsymbol{S}_B$ (see Equation 5.9).

$$\boldsymbol{S}_W = \sum_{i=1}^{c} \sum_{\boldsymbol{x} \in D_i} (\boldsymbol{x} - \boldsymbol{m}_i)(\boldsymbol{x} - \boldsymbol{m}_i)^t \tag{5.8}$$

$$\boldsymbol{S}_B = \sum_{i=1}^{c} n_i(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^t \tag{5.9}$$

In Equation 5.8 and 5.9, n_i is the number of feature vectors belonging to class i, $\boldsymbol{m}_i$ is the mean vector of the feature vectors belonging to class i, $\boldsymbol{m}$ is the mean vector of all feature vectors, and D_i is the subset of the training set belonging to class i. In order to maximize the between-class scatter and minimize the within-class scatter of $\boldsymbol{y}$, the matrix $\boldsymbol{W}$ is found by the following two operations. First solve the characteristic polynomial shown in Equation 5.10, then solve Equation 5.11 for the vectors $\boldsymbol{w}_i$. This will lead to a maximum of $c-1$ nonzero eigenvalues λ_i, and their corresponding weight vectors $\boldsymbol{w}_i$ defines the matrix $\boldsymbol{W}$. These equations have been taken from the book *Pattern Classification* [13], and a more complete walkthrough of the method and the equations can be found there.

$$|\boldsymbol{S}_B - \lambda_i \boldsymbol{S}_W| = 0 \tag{5.10}$$

$$(\boldsymbol{S}_B - \lambda_i \boldsymbol{S}_W)\boldsymbol{w}_i = 0 \tag{5.11}$$

Software based on these equations was developed in Matlab. A feature set consisting of vectors from all classes was imported and organized in a cell structure X_class of dimension $c = 6$ (each cell containing the feature vectors belonging to the corresponding class, can be thought of as D_i), to allow easy classwise access to the feature vectors. The feature set was scaled by the earlier mentioned scaling vector, see Equation 5.1. The matrices $\boldsymbol{S}_W$ and $\boldsymbol{S}_B$ were derived by straight forward numerical calculations before the matrix $\boldsymbol{W}$ was derived with the code seen in Program Code 4. First, the coefficients defining the characteristic polynomial of Equation 5.10 is found and stored in the vector p. Secondly, the roots of the polynomial are found and stored in the vector *lambda*. Finally, for each of the nonzero

```
%Define characteristic polynomial=========================
syms lambda;
[p,~]=coeffs(det(S_B-lambda*S_W));
p=double(p); %convert to numeric

%find the roots============================================
lambda=roots(p);

%Solve for W===============================================
W=zeros(length(featuresX),size(X_class,1)-1);
for i=1:size(X_class,1)-1,
    W(:,i)=null((S_B-lambda(i)*S_W),zeros(size(S_B,1),1));
end
%==========================================================
```

Program Code 4: Code to find the transformation matrix $\boldsymbol{W}$ as part of Multiple Discriminant Analysis.

values in *lambda*, Equation 5.11 is solved for $\boldsymbol{w}_i$ and stored in the matrix $\boldsymbol{W}$ as columns. Using this matrix a new feature set was derived by the code seen in Program Code 5:

```
%%Transform the feature set X to the new set Y=================
Y_class=cell(size(X_class,1),1);

for i=1:size(X_class,1),
    Y_class{i}=[X_class{i}(:,1),(W'*X_class{i}(:,featuresX)')'];
end
%==============================================================
```

Program Code 5: Code to transform the original feature set, X, into the new feature set, Y, resulting from Multiple Discriminant Analysis.

The new set was called Y_class and was declared to be a cell structure of six cells, one for each class (same structure as X_class). The first element in each vector is the class index which is used when evaluating the classifiers. This element is, therefore, not included by the transformation and is simply copied from X_class to Y_class. The rest of the vectors contain the actual features and have been transformed by applying the $\boldsymbol{W}$ matrix. The dimensions of the vectors in the new feature set have now been re-

duced from the original 17 to $c-1 = 5$. However, the ability to identify the different features as certain physical properties has been lost, as the new features are weighed sums of the original 17 features. Having implemented the transformation it has hopefully transformed the data into a new, five dimensional, feature space with good spread between the classes, and thus making the feature set more eligible for classification. As the feature space has five dimensions it is hard to show the complete picture in one plot. Instead the vectors in the new feature set are plotted in two three dimensional plots as seen in Figure 5.3a and 5.3b. By visual inspection of these two plots the classes seem to be separated into well defined regions in the new feature space. The Multiple Discriminant Analysis does not perform any form of classification, but it might be a good preprocessing step before training a classifier.

5.3 Minimum-error-rate, after performing *MDA*

The resulting feature space after performing *Multiple Discriminant Analysis* seems to be more eligible to be described by a normal distribution than the original feature space. The Minimum-error-rate classifier was therefore trained and tested using features in the new feature space. With a five dimensional feature space this results in a total number of 31 possible combinations of features, as can be derived by Equation 5.3. Again, for all feature combinations the classifier was trained and tested for each number of features being used. Some minor changes were done to the code training and testing the classifier as the vector $\boldsymbol{\mu}_i$ no longer was a function of temperature. However, the code is not repeated here, but can be found by following the path in Appendix B. The results of the best combination were saved and printed to a file. First the same set of feature data was used for both training and testing to see if the features could be described by a normal distribution. The output can be seen in Appendix A.2.1. The error

rate, P_{error}, as a function of how many features were used by the classifier, can be seen in Figure 5.4. This time the error rate decreases when adding more features. The lowest error rate was achieved when using four features with only 0.21% erroneous classifications. This implies that the features in the new feature space are well represented by the normal distribution. As a consequence of the promising results the classifier was tested with a new test set being different from the training set. The output of this test can be seen in Appendix A.2.2, and the error rates have been plotted in Figure 5.5. The results are almost as good as when training and testing with the same data. Again the lowest error rate was achieved by using four features with a P_{error} of 0.57%. Using five features resulted in the same error rate. From the plots it seems that using four features is the ideal choice, as adding the fifth feature has little or no effect on the performance of the classifier.

5.4 k-Nearest-neighbor, after performing *MDA*

Using k-Nearest-neighbor classification is useful as no assumptions needs to be made about the distribution of the feature vectors. The result obtained when using the k-Nearest-neighbor classifier will, therefore, often be as good as what is possible with the available features. Because the method is very computational expensive it was not practically possible to preform a systematic testing of all possible permutations using all 17 features. Therefore, a systematic test was performed on the feature space obtained after performing MDA. The software used to test this method is based on the Minimum-error-rate software but changing the decision rule to the kNN-decision rule with the function seen in Program Code 6.
No training of classifier parameters was necessary as the training set is used directly by the decision rule. The number of neighbors, k, used to classify was chosen to be 20. The output can be seen in Appendix A.3. The P_{error}

```
function [ output ] = NclosNeighbor(x, Y, N)
%Finds N closest neightbors to x in the set Y and returns the most frequent
%class index of those neighbors.
  min_distance=zeros(N,1);
  class_index=zeros(N,1);
  for i=1:N,
    min_distance(i)=norm(x-Y(i,2:end)');
    class_index(i)=Y(i,1);
  end
  [current_max, current_max_index]=max(min_distance);

  for i=4:size(Y,1),
    temp = norm(x-Y(i,2:end)');
    if temp<current_max,
      min_distance(current_max_index) = temp;
      class_index(current_max_index)=Y(i,1);
      [current_max, current_max_index]=max(min_distance);
    end
  end
  output= mode(class_index);
end
```

Program Code 6: Discriminant function used by the kNN classifier.

for each number of features used can be seen in Figure 5.6. The lowest P_{error} was achieved using three, four and five features for classification with an error rate at 0.12%.

5.5 Least-mean-square, after performing *MDA*

As an alternative method to the Minimum-error-rate classifier, Least-mean-square classification was tested. Again most of the software could be reused but the training and test code for the classifier needed to be changed into what can be seen in Program Code 7 and Program Code 8 respectively.

The code for training of the classifier simply generates the B and Y matrices as described by Equation 2.9 and 2.11. Finally, the A matrix is calculated by applying Equation 2.12. Furthermore, the decision rule is defined by the function *g_lms*. Using this decision rule, the code testing

```
%Training of the classifier=====================================
B=[];
Y=[];
for i=1:size(Y_class,1),
  B_i=zeros(size(Y_class{i},1), size(Y_class,1));
  B_i(:,i)=1;
  B=[B; B_i];
  Y_i=ones(size(Y_class{i},1), length(features_used)+1);
  Y_i(:,2:end)=Y_class{i}(:,features_used);
  Y=[Y;Y_i];
end
 A=(Y'*Y)\(Y'*B);
 %Desision rule: When [v, i]=g_lms(y), choose class i.
 g_lms = @(y) max(A'*y);
%================================================================
```

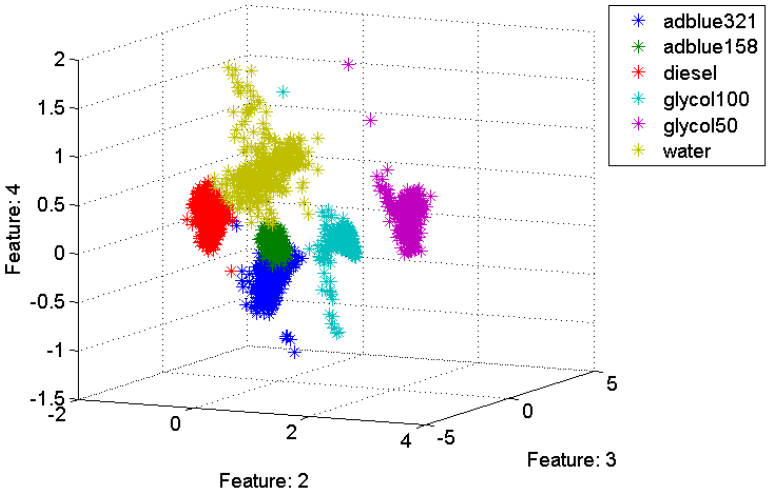
Program Code 7: Training of the Least-mean-square classifier.

the classifier was implemented in the same way as for the other classifiers.

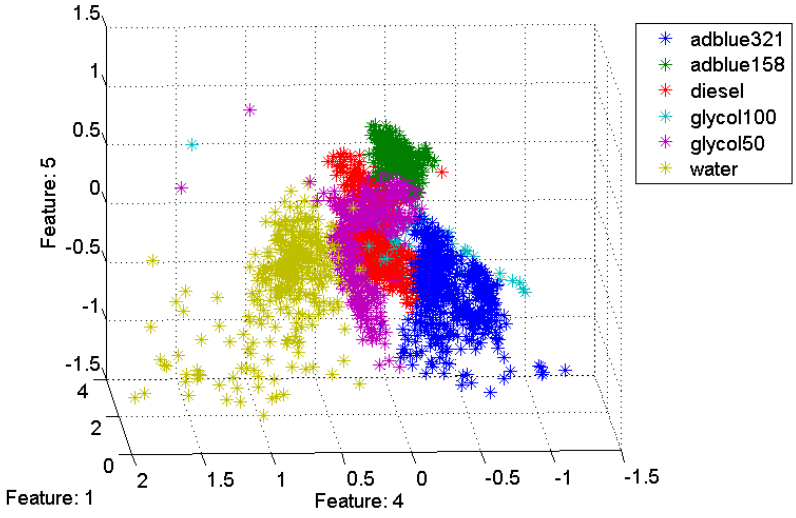
```
%initialize confusion matrix====================================
c=zeros(size(Y_class,1));
%perform test of classifier
for i=1:size(Y_test,1),
  [classification_value, classification_index]...
  =g_lms([1,Y_test(i,features_used)]');
  c(Y_test(i,1),classification_index)...
  =c(Y_test(i,1),classification_index)+1;
end
%Calculate error rate===========================================
P_error=sum(sum(c.*(ones(size(Y_class,1))...
-eye(size(Y_class,1))))/sum(sum(c));
%================================================================
```

Program Code 8: Test code for the Least-mean-square classifier.

The output can be seen in Appendix A.4, and the P_{error} for each number of features used can be seen in Figure 5.7. As the figure shows, the error rate decreases as the number of features used increases. Consequently the lowest error rate was achieved by using all five features with a P_{error} of 5.72%.



(a) Good spread between Diesel, Water Glycol100 and Glycol50. Adblue321 and Adblue158 with some overlap.



(b) Good spread between Adblue321 and Adblue158.

Figure 5.3: Three dimensional plots of points in the five dimensional feature space after performing Multiple Discriminant Analysis.

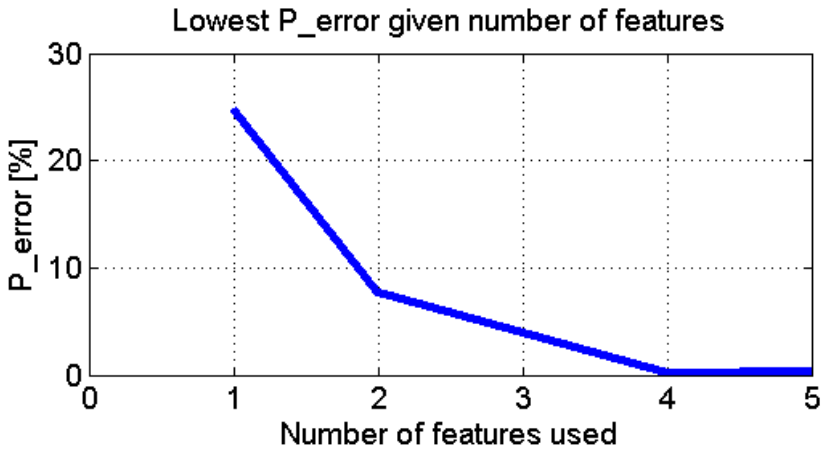


Figure 5.4: Minimum error rates P_{error} achieved by the Minimum-error-rate classifier after performing MDA on the feature set. 1 to 5 features used. Training and testing with same data set.

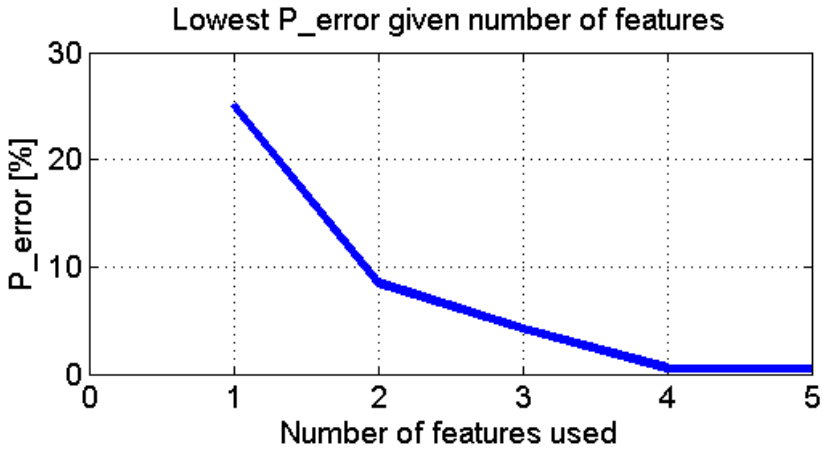


Figure 5.5: Minimum error rates P_{error} achieved by the Minimum-error-rate classifier after performing MDA on the feature set. 1 to 5 features used. Training and testing with separate data sets.

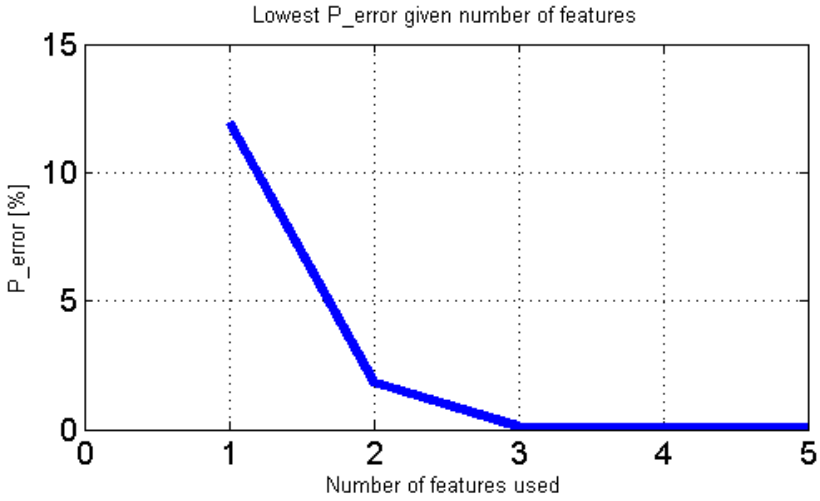


Figure 5.6: Minimum error rates P_{error} achieved by the kNN classifier after performing MDA on the feature set, k=20. 1 to 5 features used.

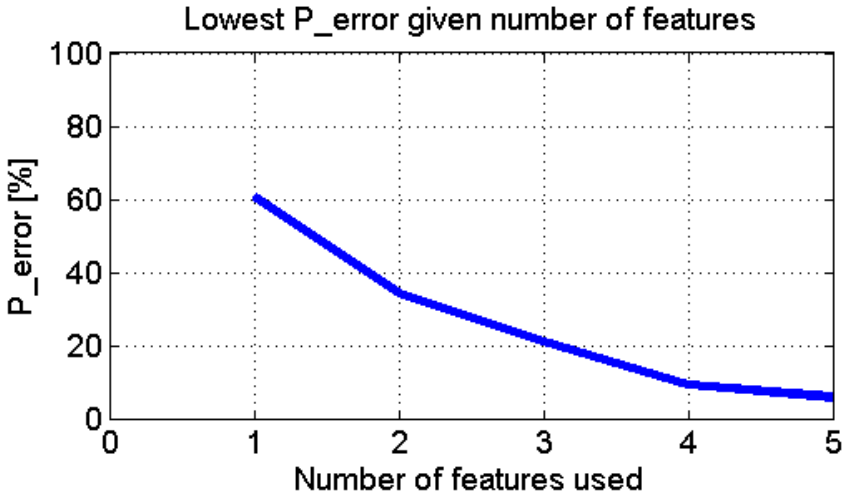


Figure 5.7: Minimum error rates P_{error} achieved by the Least-mean-square classifier after performing MDA on the feature set. 1 to 5 features used.

Chapter 6

Discussion & Future work

6.1 Discussion

The main goal of this master thesis was to develop a classifier able to cope with temperature changes and noise, typically in the form of bubbles. The work was a continuation of a project where classification under stable noiseless conditions at room temperature was achieved. In order to develop the wanted system a variety of tasks has been performed extending from circuit design, some mechanical design to programming. Every part of the system has been developed to be able to cope with the new challenge of changing temperature. Although the same theoretical basis has been used for the classifiers, as was used in the former project, implementation has been redone to fit the new requirements. The introduction of Multiple Discriminant Analysis in the system turned out crucial to the results of the classifiers.

The first implementation of the minimum-error-rate classifier using the original features space did not manage to take advantage of the new features introduced to the system. On the contrary it preformed best when only using RTT to perform classification. The main reason for the clas-

sifier not getting better from adding more features is believed to be due to an overly simplified model, assuming that only RTT is dependent of temperature.

By performing Multiple Discriminant Analysis a new feature space with only five dimensions was achieved. After finding the transformation matrix maximizing the between-class scatter and minimizing the within class scatter, it was hoped that classification would be easier as the effect of temperature on the other features no longer needed to be compensated for. As these effects would be accounted for by the transformation. With the new feature space all three classification strategies tested performed rather well, although the Least-mean-square method performed slightly worse than the two others. As the nearest neighbor classifier is very computationally heavy it is not recommended to be used as part of a finished system as long as other methods perform equally well. As a consequence of this the Minimum-error-rate classifier can be said to be the best choice of classifier given the results seen in this report.

A second goal of this thesis was to compare the classification results obtained by only using sensors currently present in the Q-sensor with what could be gained by adding new sensors. The only sensor that was added was the light absorption sensor. As it turned out to be difficult to achieve satisfying measurements with the developed prototype, the work effort was concentrated towards only using the currently used sensors. No classification results based on light absorption have been made. Instead comments were made about the difficulties met when trying out this measurement principle. Both bubbles and coating forming on the PET tubes had a significant impact on the sensor output. Also bubbles floating freely in the fluids had an impact on the result, although this could probably be filtered out by a low pass filter. It is still believed that the measuring principle can achieve good results in nice stable conditions in a lab. As part of a system performing classification on an operational vehicle it might be use-

ful to detect fluids with clearly different light absorption properties than Adblue.

6.2 Future work

In Section 4.7 it was observed that the combined performance of the new IRD algorithm and the old algorithm based on zero crossing was significantly better than their separate performance. However, finding a good way to combine the two algorithms was not pursued in this report. This is suggested as a further study.

As the tank environment a finished system will work in can be considered to be an uncontrolled environment, there are limitless possibilities as to which fluids the system might meet. In this view, limiting the classifier to distinguish between a certain number of specific fluids might not be optimal. Multivariate Calibration is suggested as a further study, as this method allows for not only distinguishing between discreet classes, but also mixtures of the different classes of fluids.

Bibliography

[1] *5.0 Round Type LED lamps, PART NO.:333IT*. Everlight. June 2004.

[2] *(5mm) High Intensity LEDs, MODEL NO.: 333-2UBC/C430*. Everlight.

[3] *(5mm) Round Shape LEDs, MODEL NO.: 333-2UBGC*. Everlight.

[4] *(5mm) Round Shape LEDs, PART NO.: 333-2UYC/S400*. Everlight.

[5] *(5mm) Round Shape LEDs, PART NO.: 333-2UYOC/S400-A7*. Everlight.

[6] *Arduino Uno*. URL: `http://arduino.cc/en/Main/ArduinoBoardUno` (visited on 05/25/2014).

[7] Tomas Bayes. *An essay towards solving a problem in the doctrine of chances*. Vol. 53. London: Philosophical Transactions of the Royal Society, 1763, pp. 370–418.

[8] *CD4051B, CD4052B, CD4053B*. Texas Instruments. Oct. 2003.

[9] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second. 605 Third Avenue, New York: WILEY-INTERSCIENCE, 2001. Chap. 1.3, pp. 9–13.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second. 605 Third Avenue, New York: WILEY-INTERSCIENCE, 2001. Chap. 2, pp. 20–83.

[11] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second. 605 Third Avenue, New York: WILEY-INTERSCIENCE, 2001. Chap. 5, pp. 215–281.

[12] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second. 605 Third Avenue, New York: WILEY-INTERSCIENCE, 2001. Chap. 10, pp. 541–542.

[13] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Second. 605 Third Avenue, New York: WILEY-INTERSCIENCE, 2001. Chap. 3.8, pp. 121–124.

[14] A.S. Dukhin and P.J. Goetz. *Characterization of Liquids, Nano- and Microparticulates, and Porous Bodies using Ultrasound*. Studies in Interface Science. Elsevier Science, 2002. Chap. 3, pp. 75–80. ISBN: 9780080543369. URL: `http://www.google.no/books?id=8UwuPmWMbE0C`.

[15] Patrick Flandrin. *Time-frequency and chirps*. URL: `http://perso.ens-lyon.fr/patrick.flandrin/SPIE01_PF.pdf` (visited on 03/25/2014).

[16] I.T. Jolliffe. *Principal Component Analysis*. 2nd ed. Springer, 2002. Chap. 1. ISBN: 0-387-95442-2.

[17] *LAMP, PART NO.: 333-2SYGC/S530-E2*. Everlight.

[18] *Light Detector Circuit*. URL: `http://www.ni.com/white-paper/13240/en/` (visited on 05/26/2014).

[19] *LMC6482 CMOS Dual Rail-To-Rail Input and Output Operational Amplifier*. National Semiconductor. Sept. 2003.

[20] Wojciech Marczak. "Water as a standard in the measurements of speed of sound in liquids". In: *the Journal of the Acoustical Society of America* 102.5 (1997), pp. 2776–2779.

[21] *NI LabVIEW Interface for Arduino Toolkit.* URL: http://sine.ni.com/nips/cds/view/p/lang/no/nid/209835 (visited on 05/25/2014).

[22] S.J. Orfanidis. *Optimum signal processing: an introduction.* 2007. Chap. 1.8, pp. 38–39. URL: http://www.ece.rutgers.edu/~orfanidi/osp2e (visited on 03/24/2014).

[23] Martina O'Toole and Dermot Diamond. "Absorbance Based Light Emitting Diode Optical Sensors and Sensing Devices". In: *Sensors* 8.4 (2008), pp. 2453–2479. ISSN: 1424-8220. DOI: 10.3390/s8042453. URL: http://www.mdpi.com/1424-8220/8/4/2453.

[24] *polyfit.* URL: http://www.mathworks.se/help/matlab/ref/polyfit.html (visited on 04/30/2014).

[25] Dr. Karl Rottmann. *Matematische Formelsammlung.* 10th ed. Spektrum forlag, 2008. Chap. 1, p. 8. ISBN: 978-82-7822-005-4.

[26] Håkon Søhoel. *project work: Classification of fluids in a closed tank.* NTNU, 2013.

[27] Dr. Kai Ming Ting. *Confusion Matrix.* URL: http://www.springerreference.com/docs/html/chapterdbid/178869.html (visited on 05/19/2014).

Appendix A

Results

A.1 Output systematic testing of Min-err

A.1.1 Minimum-error-rate using 1 to 16 features

```
1 ====================================
2 Feature(s): 10: RTT
3 817 1 0 1 105 2
4 3 830 1 1 0 6
5 0 0 936 0 0 21
6 8 0 0 992 0 0
7 86  104 0 2 551 1
8 0 0 0 0 0 592
9 P_error: 0.067589
10 ====================================
11 ====================================
12 Feature(s): 3: Conductivity Pin 4
13 Feature(s): 10: RTT
14 924 0 2 0 0 0
15 3 830 0 1 0 7
16 0 0 957 0 0 0
17 172 0 0 825 0 3
18 17  0 0 0 521 206
19 0 0 0 0 0 592
20 P_error: 0.081225
21 ====================================
```

```
22 ====================================
23 Feature(s): 3: Conductivity Pin 4
24 Feature(s): 10: RTT
25 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
26 923 0 2 0 1 0
27 9 824 0 1 0 7
28 0 0 957 0 0 0
29 247 0 0 750 0 3
30 21  0 0 0 519 204
31 0 0 0 0 0 592
32 P_error: 0.097826
33 ====================================
34 ====================================
35 Feature(s): 3: Conductivity Pin 4
36 Feature(s): 4: Conductivity Pin 3
37 Feature(s): 10: RTT
38 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
39 924 0 2 0 0 0
40 0 834 0 1 0 6
41 0 0 957 0 0 0
42 208 2 0 714 0 76
43 0 4 0 0 650 90
44 0 1 0 0 0 591
45 P_error: 0.077075
46 ====================================
47 ====================================
48 Feature(s): 3: Conductivity Pin 4
49 Feature(s): 4: Conductivity Pin 3
50 Feature(s): 5: Conductivity Pin 2
51 Feature(s): 10: RTT
52 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
53 912 12  2 0 0 0
54 0 834 0 0 1 6
55 0 0 957 0 0 0
56 6 306 0 615 1 72
57 0 5 0 0 603 136
58 0 1 0 0 0 591
59 P_error: 0.1083
60 ====================================
61 ====================================
62 Feature(s): 3: Conductivity Pin 4
63 Feature(s): 4: Conductivity Pin 3
64 Feature(s): 8: Startup noise peak strength Peak 3,
```

```
Feature(s): 10: RTT
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 18: Signal variance after startup noise
926 0 0 0 0 0
287 544 0 0 3 7
0 0 957 0 0 0
30  0 0 938 0 32
0 2 0 4 660 78
61  0 0 1 0 530
P_error: 0.099802
====================================
====================================
Feature(s): 3: Conductivity Pin 4
Feature(s): 5: Conductivity Pin 2
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 10: RTT
Feature(s): 11: Max amplitude echo1
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 18: Signal variance after startup noise
925 0 0 0 0 1
174 580 0 0 2 85
1 0 956 0 0 0
10  4 0 946 0 40
22  0 0 7 600 115
55  0 0 1 0 536
P_error: 0.10217
====================================
====================================
Feature(s): 3: Conductivity Pin 4
Feature(s): 4: Conductivity Pin 3
Feature(s): 5: Conductivity Pin 2
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
Feature(s): 10: RTT
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 18: Signal variance after startup noise
926 0 0 0 0 0
299 526 0 0 2 14
0 0 957 0 0 0
8 14  0 937 0 41
0 5 0 0 692 47
92  0 1 1 0 498
P_error: 0.10356
```

```
108 ====================================
109 ====================================
110 Feature(s): 3: Conductivity Pin 4
111 Feature(s): 4: Conductivity Pin 3
112 Feature(s): 6: Startup noise peak strength Peak 1,
113 Feature(s): 8: Startup noise peak strength Peak 3,
114 Feature(s): 9: Startup noise peak strength Peak 4,
115 Feature(s): 10: RTT
116 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
117 Feature(s): 17: Signal variance
118 Feature(s): 18: Signal variance after startup noise
119 922 0 0 4 0 0
120 317 498 0 0 2 24
121 0 0 957 0 0 0
122 26  4 0 934 0 36
123 1 0 0 7 693 43
124 70  0 0 1 0 521
125 P_error: 0.10573
126 ====================================
127 ====================================
128 Feature(s): 3: Conductivity Pin 4
129 Feature(s): 4: Conductivity Pin 3
130 Feature(s): 5: Conductivity Pin 2
131 Feature(s): 6: Startup noise peak strength Peak 1,
132 Feature(s): 8: Startup noise peak strength Peak 3,
133 Feature(s): 9: Startup noise peak strength Peak 4,
134 Feature(s): 10: RTT
135 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
136 Feature(s): 17: Signal variance
137 Feature(s): 18: Signal variance after startup noise
138 926 0 0 0 0 0
139 318 499 0 0 7 17
140 0 0 957 0 0 0
141 0 16  0 943 0 41
142 1 0 0 0 669 74
143 65  0 1 1 0 525
144 P_error: 0.10692
145 ====================================
146 ====================================
147 Feature(s): 3: Conductivity Pin 4
148 Feature(s): 4: Conductivity Pin 3
149 Feature(s): 5: Conductivity Pin 2
150 Feature(s): 6: Startup noise peak strength Peak 1,
```

```
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
Feature(s): 10: RTT
Feature(s): 14: Max peak frequency spectrum
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 17: Signal variance
Feature(s): 18: Signal variance after startup noise
926 0 0 0 0 0
311 500 0 0 7 23
0 0 957 0 0 0
0 18  0 938 0 44
1 0 0 0 599 144
63  0 1 1 0 527
P_error: 0.12115
====================================
====================================
Feature(s): 3: Conductivity Pin 4
Feature(s): 4: Conductivity Pin 3
Feature(s): 5: Conductivity Pin 2
Feature(s): 6: Startup noise peak strength Peak 1,
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
Feature(s): 10: RTT
Feature(s): 13: Damping factor between echo1 and echo2
Feature(s): 15: Factor between 1 MHz and 600KHz echo1
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 17: Signal variance
Feature(s): 18: Signal variance after startup noise
926 0 0 0 0 0
275 503 0 0 1 62
5 0 952 0 0 0
1 19  0 790 0 190
2 0 0 0 656 86
49  0 1 1 0 541
P_error: 0.13676
====================================
====================================
Feature(s): 3: Conductivity Pin 4
Feature(s): 4: Conductivity Pin 3
Feature(s): 5: Conductivity Pin 2
Feature(s): 6: Startup noise peak strength Peak 1,
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
```

```
194 Feature(s): 10: RTT
195 Feature(s): 12: Max amplitude echo2
196 Feature(s): 14: Max peak frequency spectrum
197 Feature(s): 15: Factor between 1 MHz and 600KHz echo1
198 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
199 Feature(s): 17: Signal variance
200 Feature(s): 18: Signal variance after startup noise
201 926 0 0 0 0 0
202 318 499 0 0 7 17
203 0 0 957 0 0 0
204 0 20  0 737 0 243
205 2 0 0 0 587 155
206 63  0 1 1 0 527
207 P_error: 0.16344
208 ====================================
209 ====================================
210 Feature(s): 3: Conductivity Pin 4
211 Feature(s): 4: Conductivity Pin 3
212 Feature(s): 5: Conductivity Pin 2
213 Feature(s): 6: Startup noise peak strength Peak 1,
214 Feature(s): 8: Startup noise peak strength Peak 3,
215 Feature(s): 9: Startup noise peak strength Peak 4,
216 Feature(s): 10: RTT
217 Feature(s): 12: Max amplitude echo2
218 Feature(s): 13: Damping factor between echo1 and echo2
219 Feature(s): 14: Max peak frequency spectrum
220 Feature(s): 15: Factor between 1 MHz and 600KHz echo1
221 Feature(s): 16: Factor between 1 MHz and 600KHz echo2
222 Feature(s): 17: Signal variance
223 Feature(s): 18: Signal variance after startup noise
224 926 0 0 0 0 0
225 286 494 0 0 0 61
226 5 0 952 0 0 0
227 0 19  0 613 0 368
228 3 0 0 0 584 157
229 58  0 0 1 0 533
230 P_error: 0.18933
231 ====================================
232 ====================================
233 Feature(s): 3: Conductivity Pin 4
234 Feature(s): 4: Conductivity Pin 3
235 Feature(s): 5: Conductivity Pin 2
236 Feature(s): 6: Startup noise peak strength Peak 1,
```

```
Feature(s): 7: Startup noise peak strength Peak 2,
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
Feature(s): 10: RTT
Feature(s): 11: Max amplitude echo1
Feature(s): 12: Max amplitude echo2
Feature(s): 13: Damping factor between echo1 and echo2
Feature(s): 15: Factor between 1 MHz and 600KHz echo1
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 17: Signal variance
Feature(s): 18: Signal variance after startup noise
926 0 0 0 0 0
268 498 0 0 0 75
1 0 956 0 0 0
0 17  0 512 0 471
0 0 0 0 550 194
58  0 0 0 0 534
P_error: 0.21423
====================================
====================================
Feature(s): 3: Conductivity Pin 4
Feature(s): 4: Conductivity Pin 3
Feature(s): 5: Conductivity Pin 2
Feature(s): 6: Startup noise peak strength Peak 1,
Feature(s): 7: Startup noise peak strength Peak 2,
Feature(s): 8: Startup noise peak strength Peak 3,
Feature(s): 9: Startup noise peak strength Peak 4,
Feature(s): 10: RTT
Feature(s): 11: Max amplitude echo1
Feature(s): 12: Max amplitude echo2
Feature(s): 13: Damping factor between echo1 and echo2
Feature(s): 14: Max peak frequency spectrum
Feature(s): 15: Factor between 1 MHz and 600KHz echo1
Feature(s): 16: Factor between 1 MHz and 600KHz echo2
Feature(s): 17: Signal variance
Feature(s): 18: Signal variance after startup noise
901 0 0 0 0 25
249 497 0 0 0 95
1 0 956 0 0 0
0 17  0 512 0 471
0 0 0 0 544 200
55  0 0 0 0 537
P_error: 0.21996
```

```
280 ====================================
```

A.2 Minimum-error-rate after performing MDA

A.2.1 Using the same feature data to train and test the classifier

```
1  ====================================
2  Feature(s): 3
3  206 27  4 16  0 0
4  44  52  0 106 0 3
5  1 0 395 0 0 0
6  26  21  3 127 0 5
7  0 0 0 0 189 0
8  50  14  1 25  0 82
9  P_error: 0.24767
10 ====================================
11 ====================================
12 Feature(s): 2
13 Feature(s): 4
14 224 29  0 0 0 0
15 4 201 0 0 0 0
16 0 0 396 0 0 0
17 1 1 0 141 39  0
18 0 0 0 27  161 1
19 0 5 0 0 0 167
20 P_error: 0.076593
21 ====================================
22 ====================================
23 Feature(s): 1
24 Feature(s): 2
25 Feature(s): 4
26 222 31  0 0 0 0
27 9 196 0 0 0 0
28 0 0 396 0 0 0
29 1 5 0 173 3 0
30 0 0 0 0 188 1
31 0 5 0 0 0 167
32 P_error: 0.03937
33 ====================================
34 ====================================
35 Feature(s): 2
```

```
Feature(s): 3
Feature(s): 4
Feature(s): 5
253 0 0 0 0 0
0 205 0 0 0 0
0 0 396 0 0 0
1 0 0 181 0 0
0 0 0 0 189 0
1 1 0 0 0 170
P_error: 0.0021475
====================================
====================================
Feature(s): 1
Feature(s): 2
Feature(s): 3
Feature(s): 4
Feature(s): 5
253 0 0 0 0 0
0 205 0 0 0 0
0 0 396 0 0 0
3 0 0 179 0 0
0 0 0 0 189 0
1 1 0 0 0 170
P_error: 0.0035791
====================================
```

A.2.2 Using separate feature data to train and test the classifier

```
====================================
Feature(s): 3
201 30  3 18  0 0
43  54  0 102 0 7
0 0 395 0 0 0
28  17  3 124 0 11
0 0 0 0 188 0
43  16  3 27  0 84
P_error: 0.25125
====================================
====================================
Feature(s): 2
Feature(s): 4
```

```
221 30  1 0 0 0
7 199 0 0 0 0
0 0 395 0 0 0
0 1 0 132 49  1
0 0 0 24  163 1
0 5 0 0 0 168
P_error: 0.085183
====================================
====================================
Feature(s): 1
Feature(s): 2
Feature(s): 4
222 29  1 0 0 0
10  196 0 0 0 0
0 0 395 0 0 0
3 3 0 173 3 1
0 0 0 0 187 1
0 8 0 0 0 165
P_error: 0.042233
====================================
====================================
Feature(s): 2
Feature(s): 3
Feature(s): 4
Feature(s): 5
249 3 0 0 0 0
0 206 0 0 0 0
0 0 395 0 0 0
4 1 0 178 0 0
0 0 0 0 188 0
0 0 0 0 0 173
P_error: 0.0057266
====================================
====================================
Feature(s): 1
Feature(s): 2
Feature(s): 3
Feature(s): 4
Feature(s): 5
249 3 0 0 0 0
0 206 0 0 0 0
0 0 395 0 0 0
4 0 0 178 0 1
```

```
0 0 0 0 188 0
0 0 0 0 0 173
P_error: 0.0057266
====================================
```

A.3 Output systematic testing kNN after performing MDA, k=20

```
====================================
Feature: 2
472 32  0 0 0 4
25  364 0 0 0 21
1 0 784 9 0 0
0 0 99  5 0 0
0 0 0 0 397 0
37  70  5 0 0 220
P_error: 0.11906
====================================
====================================
Feature: 1
Feature: 2
486 18  0 1 0 3
24  386 0 0 0 0
0 0 794 0 0 0
0 0 0 104 0 0
0 0 0 0 397 0
0 0 0 0 0 332
P_error: 0.018075
====================================
====================================
Feature: 1
Feature: 2
Feature: 4
505 0 0 0 0 3
0 410 0 0 0 0
0 0 794 0 0 0
0 0 0 104 0 0
0 0 0 0 397 0
0 0 0 0 0 332
P_error: 0.0011788
====================================
```

```
34 ====================================
35 Feature: 1
36 Feature: 2
37 Feature: 4
38 Feature: 5
39 505 0 0 0 0 3
40 0 410 0 0 0 0
41 0 0 794 0 0 0
42 0 0 0 104 0 0
43 0 0 0 0 397 0
44 0 0 0 0 0 332
45 P_error: 0.0011788
46 ====================================
47 ====================================
48 Feature: 1
49 Feature: 2
50 Feature: 3
51 Feature: 4
52 Feature: 5
53 505 0 0 0 0 3
54 0 410 0 0 0 0
55 0 0 794 0 0 0
56 0 0 0 104 0 0
57 0 0 0 0 397 0
58 0 0 0 0 0 332
59 P_error: 0.0011788
60 ====================================
```

A.4 Output systematic testing Least-mean-square after performing MDA

```
1 ====================================
2 Feature: 4
3 462 0 0 0 0 1
4 153 0 173 0 0 87
5 103 0 204 0 0 156
6 272 0 82  0 0 88
7 252 0 52  0 0 36
8 5 0 5 0 0 281
9 P_error: 0.60738
10 ====================================
```

```
11 ====================================
12 Feature: 3
13 Feature: 4
14 462 0 0 0 0 1
15 140 0 162 3 53  55
16 52  0 315 95  0 1
17 97  0 114 214 1 16
18 5 0 1 0 332 2
19 3 0 5 0 22  261
20 P_error: 0.34328
21 ====================================
22 ====================================
23 Feature: 1
24 Feature: 3
25 Feature: 4
26 454 0 9 0 0 0
27 106 0 192 103 0 12
28 0 0 463 0 0 0
29 25  0 2 408 0 7
30 5 0 0 1 332 2
31 0 0 0 20  25  246
32 P_error: 0.21103
33 ====================================
34 ====================================
35 Feature: 1
36 Feature: 3
37 Feature: 4
38 Feature: 5
39 459 1 1 2 0 0
40 51  286 23  50  0 3
41 0 0 463 0 0 0
42 12  36  0 388 0 6
43 5 0 0 1 332 2
44 1 4 1 13  10  262
45 P_error: 0.09204
46 ====================================
47 ====================================
48 Feature: 1
49 Feature: 2
50 Feature: 3
51 Feature: 4
52 Feature: 5
53 460 2 1 0 0 0
```

```
54 64  347 0 1 0 1
55 0 0 463 0 0 0
56 11  34  0 393 0 4
57 3 0 0 2 335 0
58 1 8 0 0 6 276
59 P_error: 0.057214
60 ==================================
```

Appendix B

Software

- path to LabVIEW software on CD: *SoftwareAppendix->LabVIEW*
- path to *CleanUpRawData.m* on CD: *SoftwareAppendix->CleanUpRawData*
- path to *Small_selection_of_files.m* on CD: *SoftwareAppendix->SmallSelectionOfFiles*
- path to *feature_extraction.m* on CD: *SoftwareAppendix->FeatureExtraction*
- path to *minimum_error_rate_systematic_testing_original_feature_space.m* on CD: *SoftwareAppendix->MinimumErrorRate*
- path to *min_err_syst_test_after_mda.m* on CD: *SoftwareAppendix->MDA->minerr*
- path to *least_mean_square_syst_test_after_mda.m* on CD: *SoftwareAppendix->MDA->lms*
- path to *kNN_syst_test_after_mda.m* on CD: *SoftwareAppendix->MDA->kNN*