Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Automated Trace Analysis of Test Traces

Akhila Manoor Lakshminarayana Bairy

Course of Study:	Information Technology (INFOTECH)
Examiner:	Prof. Dr. rer. nat. Stefan Wagner
Supervisors:	Wolfgang Fechner Jens Kölnlein (Daimler AG)
Commenced:	April 25, 2019
Completed:	October 25, 2019

Abstract

In current development of Parking Driver Assistant System, function developers get a lot of traces from various testing instances. Very often the issue is already known, and it is more or less a known standard error in the related SW-release but nevertheless the function developers have to do a time wasting trace analysis. This should be done automatically in future. So when function developers get a new trace there should be a filter which detects if it is a known error or if not.

There are 3 main parts for this topic:

1. **Trace analyser:** The traces obtained from the bussystems like 'Ethernet' and 'Flexray' should be filtered to separate the known errors. One approach is to use CANOe tool with CAPL programming language. The other approach is pattern recognition. The known errors found using the pattern recognition approach are stored in a database.

2. **User Interface:** The user interface has been developed in Python.

3. **System administration:** The system administrator can train the system for new "standard errors" in an easy way. The intention is not to bring new errors by reprogramming the complete tool but to have an interface for learning new bugs.

This is a new filtering method which is not developed until now; not that we are aware of.

Project is integrated in an overall project for testing strategy in driver assistant systems at Daimler AG in Sindelfingen together with Mercedes-Benz R&D India.

Acknowledgements

No work is complete without due recognition being given to persons who made it possible. This thesis is no exception. I would like to place on record, profound gratitude for those who have mattered the most in the successful completion of the thesis.

Firstly, I would like to express my sincere gratitude to Mr. Jens Kölnlein for giving me the opportunity to work on this thesis topic and for his continuous feedback and sparing his valuable time for discussion. Also I would like to thank Prof. Stefan Wagner and Mr. Wolfgang Fechner for supervising my thesis and for their support and guidance.

Last but not the least I would like to thank Constantin Blessing without whose constant brainstorming sessions and moral support, this thesis could not have been completed.

Contents

1 Introduction **13**
- 1.1 Motivation . . . 13
- 1.2 Goals . . . 14
- 1.3 Structure of the work . . . 15

2 Background **17**
- 2.1 File formats . . . 17
- 2.2 Pattern Recognition . . . 18
- 2.3 Databases . . . 20

3 Implementation **24**
- 3.1 Automated Analysis of Traces . . . 24
- 3.2 First Approach - Hard-coded . . . 25
- 3.3 Second Approach - Pattern Recognition . . . 29
  - 3.3.1 Discrete Signal Pattern Recognition . . . 30
  - 3.3.2 Continuous Signal Pattern Recognition . . . 33
- 3.4 Automated Trace Analysis Tool . . . 37
  - 3.4.1 Activity diagram . . . 37
  - 3.4.2 Graphic User Interface . . . 40
- 3.5 Database . . . 42
  - 3.5.1 Reading from the Database . . . 44
  - 3.5.2 Writing to the Database . . . 45

4 Evaluation **50**
- 4.1 Evaluation of 'Hard-coded' Approach . . . 50
- 4.2 Evaluation of 'Pattern Recognition' Approach . . . 50

5 Conclusion and Future Aspects **53**
5.1 Conclusion 53
5.2 Future Aspects 54

List of Figures

1.1 Existing process of detecting and fixing of errors 13
1.2 Updated process of detecting and fixing of errors 14

3.1 Discrete Signal Example 25
3.2 Discrete Signal Example with Latency 26
3.3 Slow EPS Malfunction - Good Case 27
3.4 Slow EPS Malfunction - Bad Case 28
3.5 Pattern Recognition 29
3.6 Discrete Signal Pattern Recognition 31
3.7 Continuous Pattern Recognition 34
3.8 Potential pattern in Continuous Pattern Recognition 35
3.9 activity diagram of Automated Trace Analysis Tool 37
3.10 GUI of the Automated Trace Analysis Tool 40
3.11 Database Tables 43
3.12 Database Table in Python 44
3.13 New Window to add the Malfunction to the Database 46
3.14 New window to select the timestamps 47

5.1 A screenshot of the CANoe GUI 54

Listings

3.1 CAPL pseudo code 26
3.2 CAPL pseudo code with latency 26
3.3 Slow EPS Malfunction pseudo code 27
3.4 Discrete Signal Pattern Recognition example list 31
3.5 Discrete Signal Pattern Recognition pseudo code 32
3.6 Continuous Signal Pattern Recognition pseudo code 35
3.7 Pseudo Code for Reading the Database 45
3.8 Pseudo Code for Writing to the Database 47

1.Introduction

1.1 Motivation

Nothing in this world is perfect. Everything has its flaws. Its the same even with machines. But everyday we work towards removal of the defects and perfecting them.

This thesis is done in collaboration with Parking Systems Department of Daimler AG. The results obtained from this thesis will help in detecting the defects which occur in cars.

Till now, the process of detecting and fixing an error occurring in a car involved the steps which are shown in the Fig 1.1.

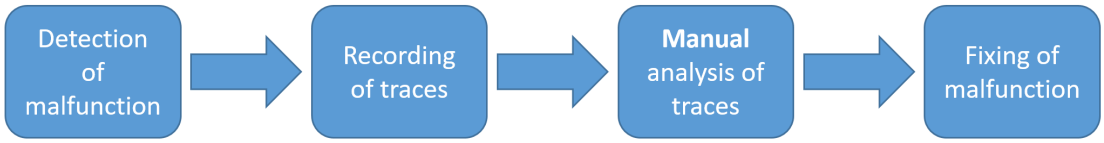

Figure 1.1: Existing process of detecting and fixing of errors

The malfunctions would be detected and then the signals would be recorded. This is in the form of a trace file which is 'Manually analysed' by someone and in the end the malfunction is fixed.

The focus of my thesis is on the 'Manual analysis' part of the above mentioned process. As a part of my thesis, the updated process would look like as shown in Fig 1.2.

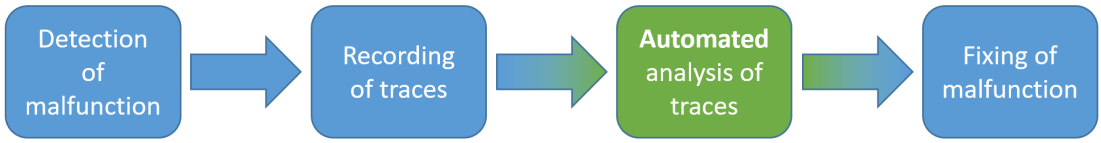

Figure 1.2: Updated process of detecting and fixing of errors

The reason for targeting the manual analysis part and automating it is given below:

1. Manual analysis is a tedious job.
2. It cannot be expected to be done by any random person. People with high competence are required.
3. Almost 80% of the traces have errors which are already known. Hence it is a waste of manpower as a high competence person is made to work on a mundane job.
4. It is also expensive.
5. There are different types of signals present in a car: Discrete signals(1 bit signal or multi-bits signal), continuous signals. Over a period of time, people tend to make mistakes even while performing a repetitive job. Therefore, manual analysis is prone to errors.

Seeing all the above mentioned reasons, we can conclude that the replacement of manual analysis of traces by automated analysis of traces is advantageous.

CANoe, a tool by Vector Informatik GmBH, is one of the tools which is used to check the signals and plot them. But till now there is no tools present which can plot the signals obtained from cars and also perform pattern recognition on them to find if known malfunctions are present in them. Hence a new tool is required which can perform the basic functions performed by CANoe and additionally also automate the process of finding the malfunctions.

1.2 Goals

The goal of this thesis is to replace the manual analysis by automated analysis. To do this I have used a method which consists of pattern recognition and a shared

database.

To choose the right kind of pattern recognition and database which suits the need of this thesis, asking the following questions will help:

1. For choosing a pattern recognition algorithm, the following questions need to be asked:
   (a) Does the algorithm work for time series data?
   (b) How many datasets does it require to perform the pattern recognition?
   (c) How long does it take to train the system?
   (d) Does the pattern recognition algorithm work for signals of long length?
2. While choosing a database, we first need to make sure the following questions can be answered:
   (a) Is it efficient to store and query time series data?
   (b) What is the read/write ratio? [1]
   (c) What kind of reads and writes are going to be performed? [1]
   (d) How big do I expect my data to be at first, and at what pace will it grow? [1]

The tool which is supposed to be developed as a result of this thesis should use the pattern recognition algorithm and database which answer all the above questions.

1.3 Structure of the work

Chapter 2: Background In this chapter we see the different methodologies which exist and their advantages and disadvantages.

Chapter 3: Implementation This chapter gives us the overview of the different implementation processes which have been used. The steps involved in the development of the tool. It also provides pseudo code of the implementation.

Chapter 4: Evaluation This chapter discusses the merits and demerits of the different approaches that have been implemented.

Chapter 5: Conclusion and Future Aspects This last chapter gives us a short conclusion about the different implementation methods used. In this chapter we also discuss about the future aspects of this thesis.

2.Background

In this section the different aspects of various parts of the topic are described. This gives an overview of various approaches studied and the reason for choosing a specific approach.

2.1 File formats

A trace file can exist in various file formats. My main focus is on the ones which are supported by Vector Informatik, GmBH. Vector supports two types of logging formats:

1. **Message based format:** This format stores the messages which are passed in the bus system along with additional information about the bus communication [2].

2. **Signal based format:** This format stores only the signal values which were present in the messages being passed in the network. In this format all the information about the message and additional information about the network are lost[2].

Message based format is of interest for me as it contains all the data about the messages being passed in the bus system. Amongst the various file formats present in message based logging format, the following ones have been investigated by me:

1. **Binary Logging File (.blf):** This format is based on 'Message-based format'. It stores the data in binary format in a very efficient way in terms of file size and read/write performance. This supports messages of all bus systems, system variables, environment variables, internal events, markers and comments[2].

   Most of the trace files which needs to be analysed using the automated trace analysis tool will be of this format.

2. **ASCII Logging File (.asc):** This format is also based on Message-based format. This format is the standard ASCII representation. This is used for data

exchange with third party programs or to include trace data in documents. This also supports messages of all bus systems, system variables, environment variables, internal events, markers and comments[2].

But ASCII is not recommended at high data rates due to its poorer performance.

3. **MDF Logging File (.mdf):** MDF stands for Measurement Data Format. This is also a binary format. This uses both message-based and signal-based logging format for reading and writing. This is the ASAM (Association for Standardisation of Automation and Measuring Systems) standard format. This format, just like BLF stores the data in an efficient way. CAN, LIN, FlexRay and Ethernet bus systems are supported[2].

   MDF is the recommended format for external data exchange or analysis with third party tools. It is more efficient when compared to BLF in some configurations.

The trace files used at Daimler AG are usually in the BLF format. By comparing the different logging formats mentioned above, we can conclude that MDF would be the ideal format to use in the 'Automated Trace Analysis Tool' as it is more efficient than BLF, is the standard ASAM format and is also the recommended format for third party applications.

2.2 Pattern Recognition

Pattern recognition can be defined as identifying patterns and regularities in data. A technical definition of pattern recognition is:

"The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories [3]"

There are various methods to perform pattern recognition. The signals on which I need to perform pattern recognition are time series signals. A few of the methods for pattern recognition in time series which I investigated as a part of my thesis are as follows:

1. **Machine Learning:** Whenever we say pattern recognition, the first thing that comes to mind is machine learning. Both of these activities can be viewed as two facets of the same field [3]. There are various methods based on the categorisation of the data using different learning procedures in which pattern

recognition can be performed using machine learning. The most common ones are given below:

(a) **Supervised learning:** Supervised learning is a process in which a system is trained to map output variables to input variables by observing some example input-output pairs [4]. For time series data, memory based reasoning methods have found to be giving fruitful results. Some of these methods are:

i. **Nearest neighbor (NN):** This is a type of classifier in which the distance between the unknown sample and other samples are computed; and is classified into the type it is closest to [5]. Instead of only one closest sample, if you consider a couple of closest samples and classify the unknown sample based on it, then it becomes "K-Nearest Neighbors (KNN)".

ii. **Decision Tree:** A decision tree is a flowchart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions [5].

(b) **Semi-Supervised learning:** In this type of learning, both labelled and unlabelled type of data is used. Most of the real world problems fall in this category. You can use unsupervised learning techniques to discover and learn the structure in the input variables [6]. You can also use unsupervised learning techniques to make best guess predictions for the unlabelled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data [6].

(c) **Unsupervised learning:** Unsupervised learning is performed on datasets which have unlabelled data. An example of this method of learning is 'Clustering'. Hierarchical clustering is the most widely used clustering approach. The advantage of this method is that the user doesn't require to provide any parameters such as the number of clusters [7]. The disadvantage of this method is that it has quadratic complexity which makes it only suitable for small datasets.

2. **Shape-based similarity:** The shape-based similarity between two time series signals can be determined by comparing local point values of the two signals [7]. There are various methods which use this shape-based similarity method

to perform pattern recognition in time series signals. The most famous ones among them are given below:

(a) **Euclidean distance:** Euclidean distance is the most commonly used method for time series signals. But the major disadvantage of this method is that it requires both input sequences to be of same length [7].

(b) **Dynamic Time Warping:** The problem faced by Euclidean distance method, both input sequences need to be of the same length, can be resolved using dynamic time warping (DTW). While DTW is a more robust distance measure than Euclidean Distance, it is also a lot more computationally intensive than Euclidean Distance, even with the presence of a global constraint [7].

(c) **Fast Dynamic Time Warping:** DTW has a quadratic time and space complexity that limits its use to only small time series datasets [8]. This complexity can be overcome by using FastDTW. FastDTW uses a multilevel approach that recursively projects a solution from a coarse resolution and refines the projected solution [8].

3. **Structural similarity:** Shape-based similarities have one major disadvantage; they work well for only short signals [7]. This drawback can be overcome by using structural similarity. In this method, similarity between long time series is measured based on higher-level structures [7].

Although machine learning seems to be the most common go-to approach, it is not suitable for this thesis. As it requires a huge amount of datasets for training and testing which is not possible to obtain in my case. Also it requires a lot of time to train the system. The other methods also have their own drawbacks. Hence for the purpose of this thesis, I've developed new algorithms to perform pattern recognition. This will be explained in the coming chapters.

2.3 Databases

Databases are part of everyday life. Before databases came into existence, everything used to be physically written on paper and stored. It used to be difficult to search or modify or remove something from the record. Databases have made everyone's life simpler. There are various types of databases:

1. **The hierarchical database model:** This was one of the first models to be developed. This was developed by IBM. In this model, there is a parent/root record which then has child records.

   The advantages of hierarchical model is that it is simple to use and is fast. But the disadvantages outweigh the advantages. The whole tree has to be traversed to access data. Also it cannot easily represent relationships between different types of data [9].

2. **The network database model:** In this model, the child record can have multiple parent records. This wasn't possible in hierarchical model.

3. **The relational model:** This is the most used model. In this model you can have tables and each table contains records. The main advantage of the relational model is that it provides consistency in the data [9].

The relational database model is the most commonly used model. The database management systems which use this model are called as "Relational Database Management System" (RDBMS). There are several databases which fall under RDBMS. A few of them are:

1. **MySQL database:** MySQL database is an open source database implementation. This is designed for speed and reliability [10]. Also in this database the security is high as it comes with a script which helps in setting the password for the root user.

   One of the main disadvantages of this is that it is dual-licensed software. Therefore some of the features and plugins might be only available for proprietary editions.Also it does not implement full SQL standard. MySQL is mostly used for web application development.

2. **PosegreSQL:** PostgreSQL is also known as Postgres. It is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads [11]. Postgres is capable of efficiently handling multiple tasks at the same time, a characteristic known as concurrency [10].

   Postgres implements most of the SQL standards. But one of the main disadvantage is that for every new client connection, this creates a new process [10].

   PostgreSQL is also mostly used for web application development. Also PostgreSQL databases support a wide range of operating systems and migrating from one to another is easy.

3. **SQLite:** SQLite is a self-contained, file-based, and fully open-source RDBMS known for its portability, reliability, and strong performance even in low-memory environments [10]. one of the main advantages of SQLite is that it is serverless [12]. Setting up an SQLite database is nearly instant, there is no server to set up, no users to define, and no permissions to concern yourself with [13].

   One of the biggest advantages of SQLite is that it is very lightweight. Also it is user friendly and doesn't require any username or password. The entire database is stored as a single file.

   Multiple people can access this database but only one person can make changes at any given time.

   SQLite is mostly used for developmental and prototyping purposes.

The above different types of RDBMS have been looked into and I felt that SQLite suits my purpose.

3.Implementation

In this chapter we'll look into the different approaches that I have taken to perform filtering of known errors from signals. After that we'll see the implementation of the 'Automated Trace Analysis Tool'.

3.1 Automated Analysis of Traces

The first and foremost question which comes to mind when you hear the term 'traces' is, 'what are traces?'.

Traces are files which contain values of signals and how they change over time. Depending on the type of trace file, message-based or signal-based, it might also contain more information regarding the signals and the bus systems. Traces are obtained from various bus systems such as CAN, LIN, FlexRay etc., which are present in the car.

The next question which arises is 'What is automated analysis of traces?'.

Automated analysis involves automated detection of erroneous patterns in one or more signals. In other words, there is no requirement for manual intervention for the detection of errors/malfunctions.

The follow-up question would be 'What does a malfunction refer to in the automotive industry?'.

Malfunction means a failure to function normally. In the automotive industry an example of a malfunction would be, turning on the left indicator but the right indicator starts to blink or no indicator blinks. These are serious malfunction and failure to rectify this could result in major accidents.

The traces which are obtained for this thesis are from the parking systems which contains the PARKMAN ECU. The traces are obtained from this ECU using FlexRay. And the format in which it is stored is BLF format (more information about this is given in section 2.1).

The traces obtained from the car need to be checked for malfunctions. In this thesis I've used two approaches which are discussed in the coming sections.

3.2 First Approach - Hard-coded

The first approach which I had implemented to detect the malfunctions was a hard-coded approach. This approach was performed in CANoe using the CAPL programming language.

CANoe is a tool by Vector Informatik GmBH for analysis of measurement data. CAPL stands for Communication Access Programming Language. It is a programming language which is developed by Vector Informatik GmBH for the automotive sector. It is similar to the programming language C. It is used in the CANoe environment for accessing and controlling vehicle network data.

To explain this approach better let us consider the example given in the Fig 3.1. In this example we can see that there are 2 signals namely Signal A and Signal B. Signal A can have values A0, A1 and A2 and Signal B can have values B0 and B1.

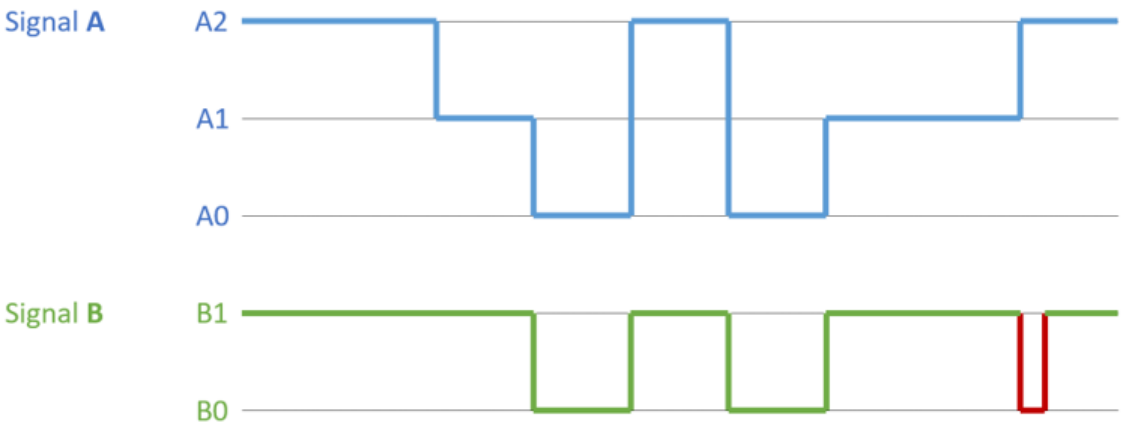

Figure 3.1: Discrete Signal Example

Whenever the Signal A is A0, Signal B should be B0 and vice versa. But in the Fig 3.1 we see that at the end of the trace, when Signal A changes from A1 to A2, Signal B instead of staying at B1 it changes to B0 and then back to B1. This would constitute a malfunction.

Listing 3.1 shows a hard-coded implementation for detecting this malfunction in pseudo code. This is a simple code. But in reality we need to account for latency as well. The example shown in Fig 3.1 with latency would look as shown in the Fig 3.2.

```
if ((Signal_A != A0 && Signal_B == B0)  ||
     (Signal_A == A0 && Signal_B != B0) )
{
  // error
}
```

Listing 3.1: CAPL pseudo code

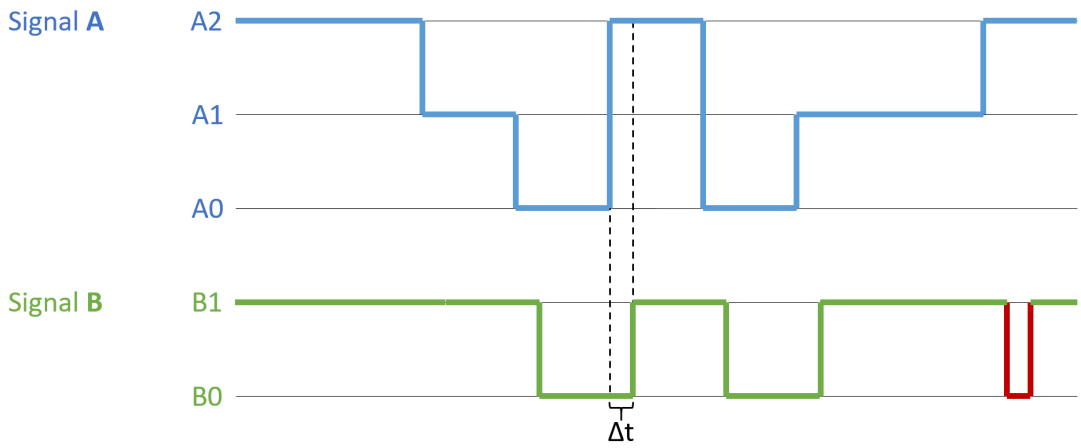

Figure 3.2: Discrete Signal Example with Latency

The latency differs from one trace to another. Hence hard-coding this as mentioned in the Listing 3.1 is not possible. For this we would need to define a threshold value. If the signals exceed the threshold value then its a malfunction. This can be explained using the pseudo code shown in Listing 3.2.

```
if (((Signal_A != A0 && Signal_B == B0)  ||
     (Signal_A == A0 && Signal_B != B0) )  &&
     (threshold > δt))
{
  // error
}
```

Listing 3.2: CAPL pseudo code with latency

Another real life example is given in the Fig 3.3 and 3.4. In this example, 'Parking Distance' signal refers to the distance between the tyre and the curb, and

'Steering Wheel Angle' signal represents the steering wheel angle at any given time. Its pseudo code is given in listing 3.3

Fig 3.3 represents how the two signals would look like in a good case scenario. The green highlighted part shows that the value change in 'Steering Wheel Angle' is minimum when 'Parking Distance' has a local maxima.

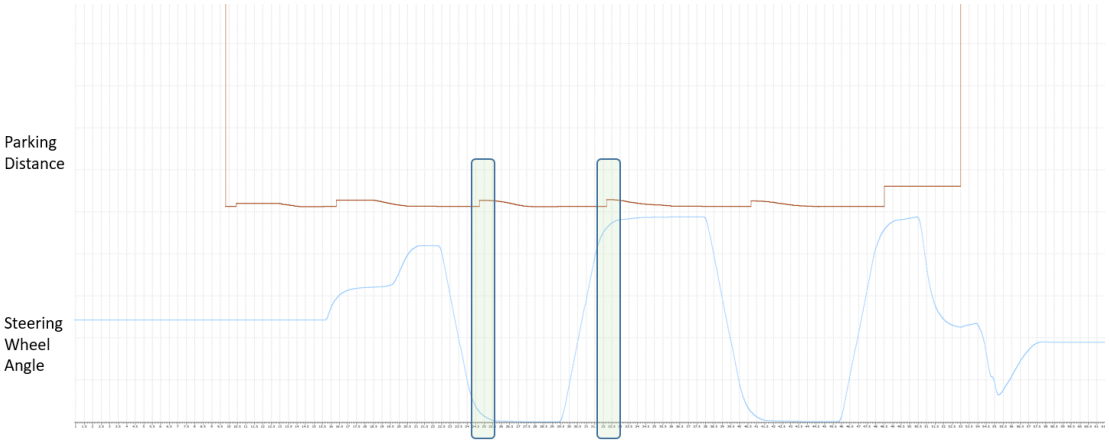

Figure 3.3: Slow EPS Malfunction - Good Case

Fig 3.4 represents how the two signals would look like in a bad case scenario. The red highlighted part shows that the value change in 'Steering Wheel Angle' is more when 'Parking Distance' has a local maxima.

```
1 if (Parking_Distance has local maxima):
2     Parking_Dist_old_value = Parking_Distance current value;
3     steering_whlAngl_1 = Steering_Wheel_Angle at the current time;
4
5     if (Parking_Distance has value 5 less than
    Parking_Dist_old_value):
6         steering_whlAngl_2 = Steering_Wheel_Angle at the current
    time;
7
8         if (difference between steering_whlAngl_1 and
    steering_whlAngl_2 is greater than threshold angle):
9             //Malfunction has occurred!
```

Listing 3.3: Slow EPS Malfunction pseudo code

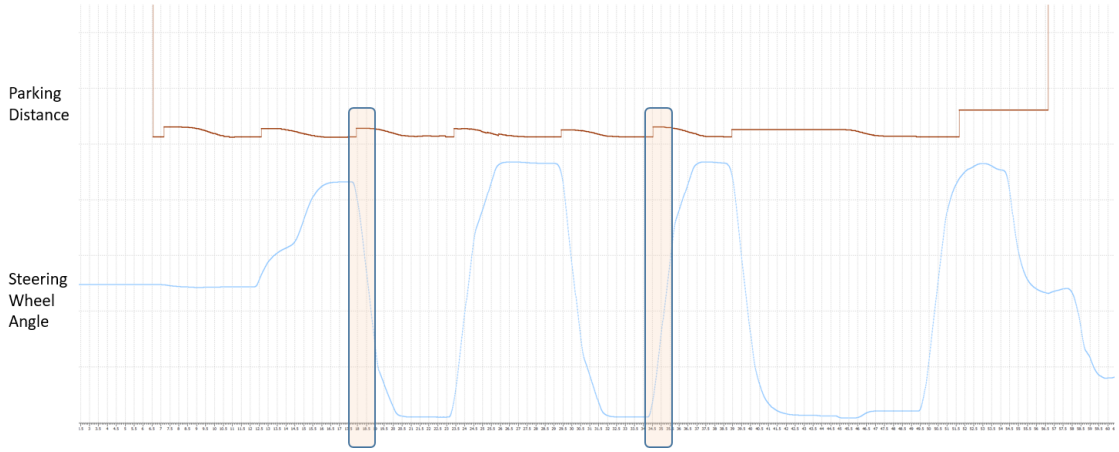

Figure 3.4: Slow EPS Malfunction - Bad Case

This approach was implemented for two malfunctions.
The merits and demerits of this approach will be discussed in the chapter 4.

3.3 Second Approach - Pattern Recognition

The other approach which I have implemented involves pattern recognition in signals. This approach was performed using the programming language Python.

We discussed the already existing methods for performing pattern recognition in the section 2.2. As discussed in that chapter, the machine learning approach is not suitable for this thesis as it requires a lot of data to train the system. The other methods are also not suitable as they are only for small samples of signals.

Hence I've come up with two new approaches to perform the pattern recognition. These approaches can be applied to signals of any lengths. Also these approaches require only one signal to mark the pattern. They don't require any training of the system.

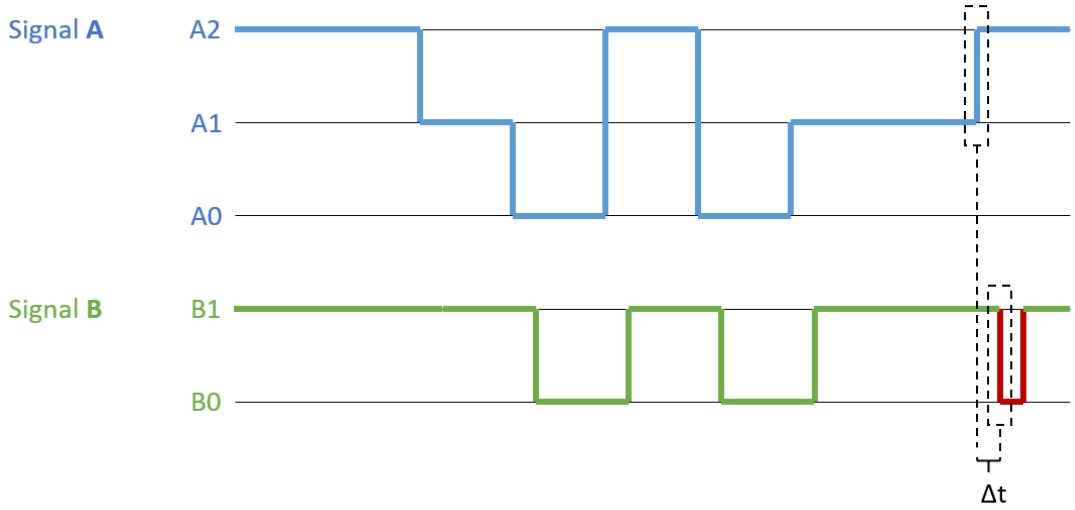

Figure 3.5: Pattern Recognition

An overview of the approaches which I've used can be given by explaining the Fig 3.5. It is same as the Fig 3.2. In the Fig 3.5 we see that where the error occurs, that pattern has been marked in both Signal A and Signal B. These patterns will then be stored in the database where all the other malfunctions would also be stored. Not just the samples, but also their timestamps will be stored. When a new trace file is uploaded, we check if Signal A and Signal B of this new file have these saved patterns in them. Also the time difference is checked. If the patterns match and are

within the time difference threshold, then it would be considered as a malfunction present in the uploaded trace file.

To give an in-depth overview of the pattern recognition algorithm, we'll have to look into each of the pattern recognition algorithm separately. There two types of pattern recognition which are:

1. Discrete Signal Pattern Recognition
2. Continuous Signal Pattern Recognition

3.3.1 Discrete Signal Pattern Recognition

Discrete signal pattern recognition is performed for signals which have strings as values on the y-axis. The steps involved in executing this are given as follows:

1. Firstly two new lists are created to store the samples and timestamps of pattern signal.
2. In the new lists the first value and its timestamp are stored.
3. Whenever there is a value change, the new value is added to the list. Along with it its timestamp is stored in the corresponding list.
4. Finally the last value in the pattern signal is appended to the lists. Even its timestamps is stored in the timestamps list.
5. The above steps are repeated for the input signal.
6. Check if the pattern samples list matches the signal samples list.
7. Find the threshold for the pattern recognition by using the time difference from the pattern timestamps.
8. Also check if the δt for the input signal is within the threshold.
9. If the steps 6 and 8 are satisfied, then a malfunction is present.

To explain this better, we refer to the Fig 3.6. In this figure we see the same signals Signal A and Signal B as before. For this if we apply discrete signal pattern recognition algorithm as mentioned above, then the output lists would be as shown in the listing 3.4.

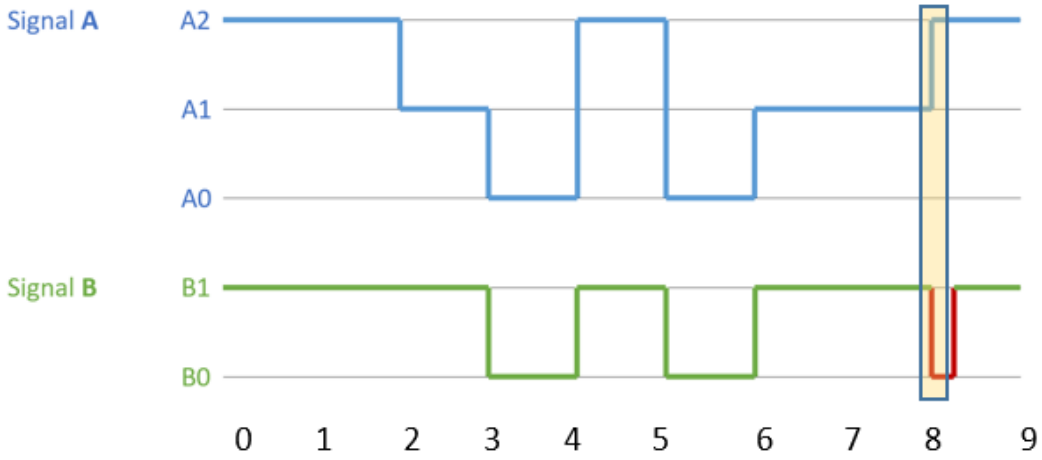

Figure 3.6: Discrete Signal Pattern Recognition

```
Signal samples_A = [A2, A1, A0, A2, A0, A1, A2, A2]
Signal timestamps_A = [0, 2, 3, 4, 5, 6, 8, 9]
Signal samples_B = [B1, B0, B1, B0, B1, B0, B1, B1]
Signal timestamps_B = [0, 3, 4, 5, 6, 8, 8.5, 9]

Pattern samples_A = [A1, A2, A2]
Pattern timestamps_A = [7.8, 8, 8.2]
Pattern samples_B = [B1, B0, B0]
Pattern timestamps_B = [7.8, 8, 8.2]
```

Listing 3.4: Discrete Signal Pattern Recognition example list

Once these lists are made then it checks if signal samples_A list contains Pattern samples_A list in it. If it does, then it checks if the time difference is similar. Once this is done, the same test is performed for signal samples_B with Pattern Samples_B. If it matches then the Signal timestamps_B and pattern timestamps_B are checked to see if the threshold matches. After this is done, the time difference between the first elements of the two pattern timestamps lists are calculated and this value is used as threshold. If the time difference between the Signal timestamps of the two signals are equal to or less than the threshold then it is printed as a malfunction.

In the above example, the patterns match in both Signal A and Signal B with the Pattern samples_A and Pattern samples_B. Also the time difference is within the threshold. Hence it is a malfunction.

The pseudo code is as given in the listing 3.5.

```
def discrete_signal_pattern_recognition(self, pattern_timestamps, pattern_samples, signal_timestamps, signal_samples):

    test_pattern_samples = []
    test_pattern_timestamps = []

    # Append the first sample, samples whenever there is a change and the last sample
    test_pattern_samples.append(pattern_samples)
    test_pattern_timestamps.append(pattern_timestamps)

    test_signal_samples = []
    test_signal_timestamps = []

    # Append the first sample, samples whenever there is a change and the last sample
    test_signal_samples.append(signal_samples)
    test_signal_timestamps.append(signal_timestamps)

    # Check if the signal_samples contain the pattern_samples
    for index_i in range(len(test_signal_samples)):
        if test_signal_samples has test_pattern_samples:
            is_match = True

            # Compute the pattern duration
            pattern_duration = difference between next and current elements of test_pattern_timestamps

            # Compute the signal duration
            signal_duration = difference between next and current elements of test_signal_timestamps

            # Minimal duration of signal
            if signal_duration is less than half of pattern_duration or greater than 1.5 times of pattern_duration:
                is_match = False
                break

        if is_match is True:
        # Malfunction found!
```

Listing 3.5: Discrete Signal Pattern Recognition pseudo code

3.3.2 Continuous Signal Pattern Recognition

In the previous subsection we saw how to perform Discrete signal pattern recognition. In this subsection we'll see more about continuous signal pattern recognition.

Continuous signal pattern recognition is implemented for signals which have real numbers as values in the y-axis. I can't take complete credit for this approach as I got the basic idea of how to perform this pattern recognition from one of the answers in stack exchange [14]. Using the method mentioned as the basis, I performed a few more operations on it to get the final result in my pattern recognition.

The steps to perform continuous signal pattern recognition are as follows:

1. Auto correlate Pattern samples to find the maximum value.
2. Perform cross-correlation between signal samples and pattern samples.
3. Filter the cross-correlation values by using the maximum value obtained in step 1 as threshold.
4. Find the start indices of the filtered cross-correlation values. Let us call these points as potential-start-indices.
5. Embed the pattern samples at the potential-start-indices.
6. Find the difference between the signal and pattern samples at the potential start indices. Let us call this difference as pattern-difference.
7. Take the variance of each pattern-difference.
8. Find the median of all the variances. This will be used as threshold value to find the patterns in the signal.
9. Choose the variances whose value is less than or equal to the median.

After reading the steps, the first question to pop into our head is 'Why is auto-correlation done using the pattern samples?'.

When a signal is cross-correlated with itself, the process is referred to as auto-correlation. In our pattern recognition method pattern samples are used as a mask. So if the input signals have high values which don't correspond to a similarity in shape with the pattern signal (mask), they will still return a high value of cross

correlation as the sum of the element-wise product is high [14]. This is where the auto-correlation comes into picture. The output of auto-correlation can be used to filter out these high values in the input signal.

Another question which can be asked is 'How is pattern recognition performed using cross-correlation?'.

Cross-correlation is nothing but convolution between one signal and inverted value of another signal. In our case, convolution is performed between the inverted pattern signal and the input signal. The places where you get a peak in the correlation are potential matches.

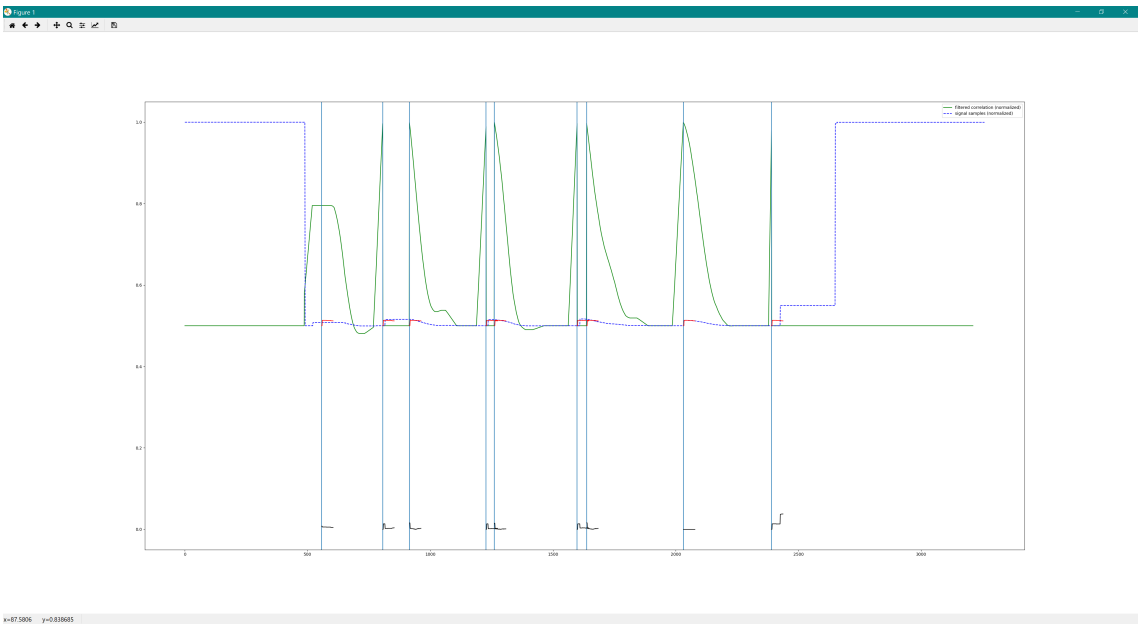

Figure 3.7: Continuous Pattern Recognition

In the Fig 3.7, the blue dotted line represents the normalised values of signal samples. The green line represents the normalised values of filtered correlation values. The blue straight lines represent the start points of potential patterns. The red lines depicts when the pattern is inserted at the start points. The black lines represent the difference between the pattern and input signal at that place (pattern-difference that was mentioned in step 6).

To have a better look at the above mentioned things, a zoomed image of Fig 3.7 is shown in Fig 3.8. In this image we can clearly see that here the pattern

samples (red line) when inserted at the start point (blue straight line) seen in this picture perfectly matches the input signal samples (blue dotted line). Hence the pattern-difference (black line) is a straight line. This would mean that variance of this pattern-difference is zero. It would thus be definitely below the mean of the variances. Thus we can conclude that the pattern recognition at this point is successful and this start point would be termed as the place where the malfunction occurs.

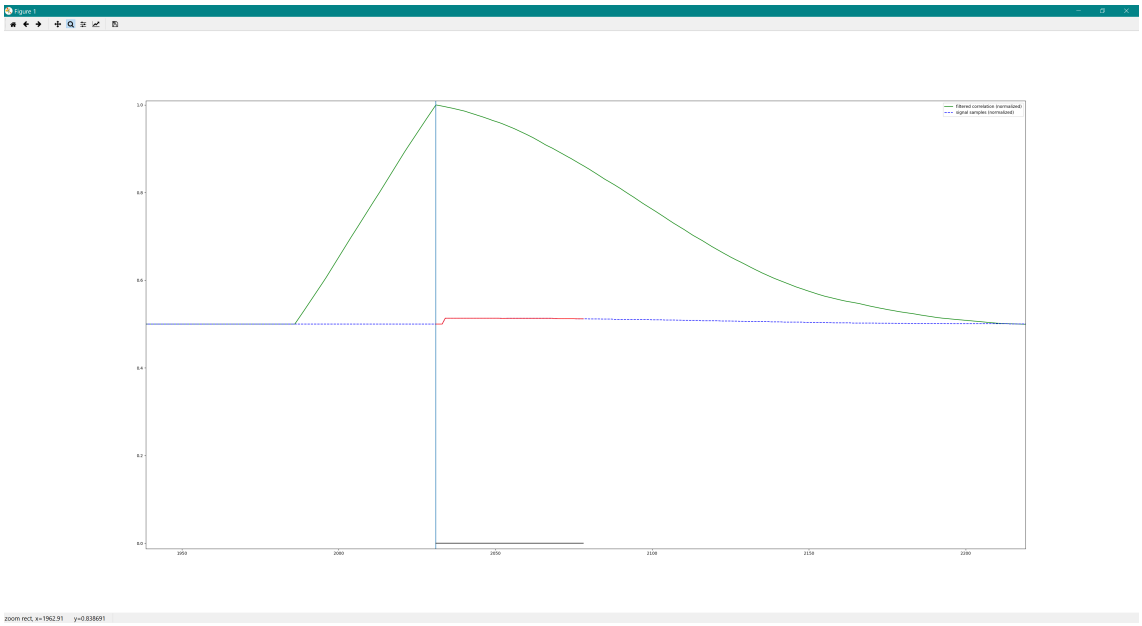

Figure 3.8: Potential pattern in Continuous Pattern Recognition

The pseudo code for continuous signal pattern recognition can be seen in the listing 3.6

```
def continuous_signal_pattern_recognition(self, pattern_timestamps,
    pattern_samples, signal_timestamps, signal_samples):
    # Auto-correlation of pattern samples
    max_pattern_correlation_value = max(np.correlate(
   pattern_samples, pattern_samples))

    # Cross-correlation between pattern samples and signal samples
    correlation = np.correlate(signal_samples, pattern_samples)

```

```
    # Find the filtered value of correlation after comparing it
with the auto-correlation value
    correlation_filtered = correlation values which are less than
or equal to max_pattern_correlation_value

    # Find the start points where a potential pattern might exist
    potential_pattern_start_indices = peak values in the
correlation_filtered which are at least 0.5 times of
max_pattern_correlation_value

    for potential_pattern_start_index in
potential_pattern_start_indices:
        # Find the pattern_difference for each
potential_pattern_start_index
        pattern_difference = difference between pattern samples and
     signal samples at potential_pattern_start_index
        # Find the variances
        variances.append(np.var(pattern_difference))

    # Find the median of the variance
    median = statistics.median(variances)

    # Find the start point where variance is less than or equal to
median
    for i in range(len(potential_pattern_start_indices)):
    if variance[i] <= median:
        indices.append(potential_pattern_start_indices[i])

    for index in indices:
        signal_timestamps_return.append(signal_timestamps[index])

    # Malfunction is found at timestamps present in
signal_timestamps_return!
```

Listing 3.6: Continuous Signal Pattern Recognition pseudo code

In this section we saw in detail the two different methods, 'discrete signal pattern recognition' and 'continuous signal pattern recognition', and how they are used to perform pattern recognition in this thesis. The merits and demerits of using pattern recognition and how it is better than the previous section, 'First approach - Hard-coded', will be discussed in chapter 4.

3.4 Automated Trace Analysis Tool

Till now in this chapter we saw the different approaches used in this thesis. The second approach which used pattern recognition was implemented using Python and then used in a tool that I developed which is called "Automated Trace Analysis" tool.

3.4.1 Activity diagram

The Automated Trace Analysis tool was developed using Python and C# programming languages. The Graphic User Interface (GUI) and majority of the functionalities of the tool has been coded using Python. C# has been using only for one functionality which will be discussed in detail in the coming part. The activity diagram of this tool is given in the Fig 3.9.

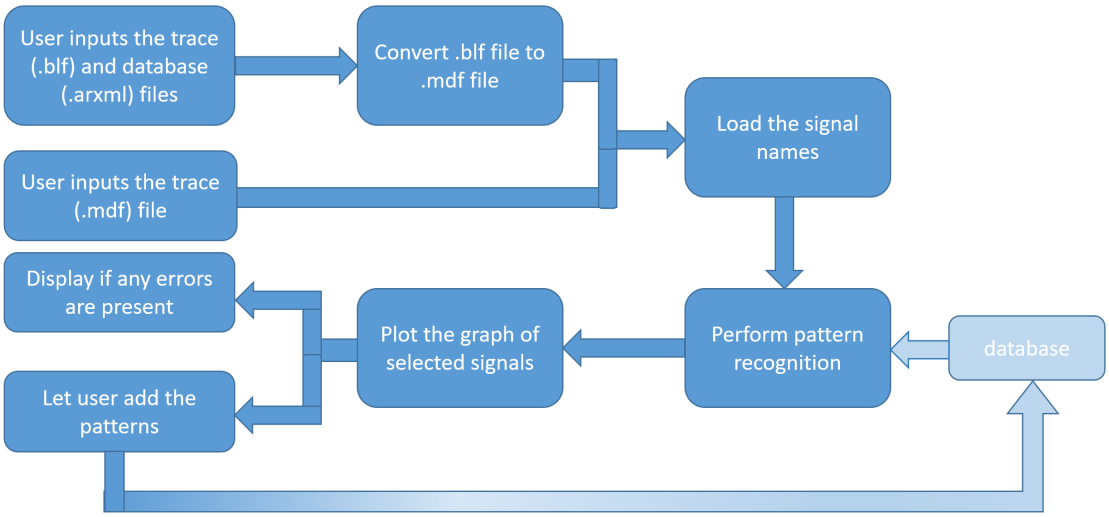

Figure 3.9: activity diagram of Automated Trace Analysis Tool

The automated trace analysis tool performs a variety of functionalities as you can see from the activity diagram. I will explain each block in detail below:

1. **User inputs the BLF trace and ARXML database files:** This would be the first step the user can perform. He must input the trace file. If the trace

file is a BLF file (more information regarding this in section 2.1), the user will be immediately prompted to choose a database file (ARXML format). This is done as the database file will contain details regarding how the signals look like in the BLF trace file.

2. **Convert BLF file to MDF file:** This step is performed if the user chooses a BLF trace file and an ARXML database file. In this step the BLF file is converted into an MDF file. This conversion is performed as Vector Informatik GmbH recommends the MDF file format to be used for third party applications. More details about this can be seen in the section 2.1.

   The conversion of BLF to MDF is performed in the background. This is done using C# language. For this conversion, the CANoe tool by Vector Informatik GmbH is called using the C# code. It is fed with the BLF trace file and the ARXML database file. Using these two files, CANoe converts the input trace file into an MDF file. This file can now be used in our tool along with open source libraries which are available as it follows the ASAM standard.

3. **User inputs the MDF trace file:** This step can be used as an alternate step for step 1. If the user inputs a MDF trace file, then there is no need for ARXML database file. Also step 2 becomes redundant as we already have the file in the required format.

   Once MDF file is received either through step 2 or 3, it directly goes to the next step.

4. **Load the signal names:** As the name suggests, in this step the MDF file is read and the names of all the signals which are present in the file are displayed on the screen in a widget called signal list view. Usually a trace file will contain over 4000 signals which are present in a car. To help the user with finding the signal which he wants, there is also a search feature present in the tool. Also at the bottom of the widget, the total number of signals present in the trace file is also displayed.

5. **Perform Pattern Recognition:** Once the MDF file is obtained, using the malfunctions present in the database, we perform pattern recognition between the signals in the database and the signals obtained from the MDF file. There are two types of pattern recognition performed based on the type of the signal: Discrete signal pattern recognition and Continuous signal pattern recognition. This is discussed at length in subsection 3.3.

6. **Plot the graph of selected signals:** In the tool, there is a signal view widget present. Whenever the user selects a signal name from the signal list view widget, a plot of that signal is draw in the signal view widget. And if the user deselects a signal name from the signal list view, then that signal plot is removed from the signal view widget.

7. **Display if any errors are present:** There is a console widget present in the tool which is used to display any malfunctions found in the uploaded trace file.

   If there are any malfunctions present, then the malfunction name, its instance name, the timestamps at which the malfunctions occurred and the signal names of the signals which are involved are displayed.

8. **Let user add the patterns:** Initially when the outline was prepared for the tool, this was not a part of it. But later on it was decided that this would make the tool more user friendly and easier to use. Hence this feature was added.

   In this step the user is given a freedom to add malfunctions to the database by choosing the signals from the signal list view. Then he can give a name for the malfunction and also an instance name. If the malfunction name already exists in the database, then just a new instance is created for the existing malfunction. Else a new malfunction is created in the database. After given the malfunction and instance names, the user can then select the timestamps for which the malfunction occurs.

9. **Database:** The detailed working of the database is discussed in the section 3.5. The database performs two main functions: read and write.

   Read function is performed as soon as the MDF file is obtained, either through steps 1 and 2 or by steps 3, by the tool. When pattern recognition is being performed the existing malfunctions are read from the database.

   Write function comes into picture when the user selects to add a pattern to the database, i.e. when step 8 is performed. Once the user is done selecting the timestamps for the signals, the signals along with the selected timestamps and samples are stored in the database.

The basic functionalities of all the blocks present in the activity diagram of the automated trace analysis tool has be explained.

We shall now look into the GUI of the Automated Trace Analysis Tool.

3.4.2 Graphic User Interface

In the previous subsection we saw the activity diagram of the Automated Trace Analysis Tool. In this chapter we go into the details of how the graphical user interface (GUI) of the tool looks like and what does each term mean.

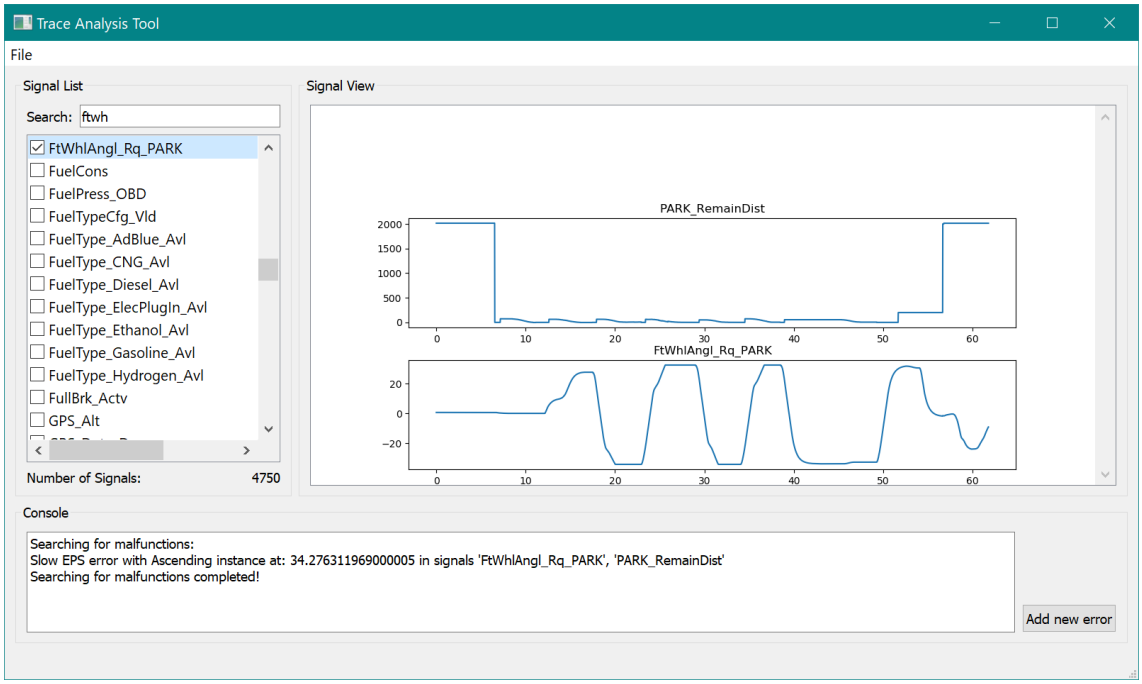

Figure 3.10: GUI of the Automated Trace Analysis Tool

The Fig 3.10 is a screenshot of the GUI of Automated Trace Analysis tool. In this tool we can see 5 main components. Those components are:

1. **'File' Menu bar:** The file menu bar has one action which is present in it. This action is called as 'Open'. It has a shortcut to it 'Ctrl + O'. When this action is clicked, it opens a File Dialog box in which you can either select a BLF trace file or an MDF trace file. If a BLF trace file is selected, then another File Dialog box opens in which you can select an ARXML database file. If BLF trace file + ARXML database file combination is uploaded, then the BLF to MDF conversion happens in the background with the help of CANoe tool.

2. **'Signal List' widget:** This widget has 3 parts:

(a) **Signal names checklist:** Once the MDF file is obtained by the tool, all the signal names are extracted from the file and displayed on this widget. It is in the form of a checklist so that the signal names can be selected to be viewed in the signal view.

(b) **Search bar:** This widget, as the name suggests, helps in searching the signal name which the user wants.

(c) **Number of signals widget:** The widget displays the number of signals which are present in the uploaded MDF file.

3. **'Signal View' widget:** Signal view widget is the place where the signals are plotted for the user. This has two functionality:

   (a) **Add signal plot:** When the user checks a signal name in the 'signal list' widget, that signal is plotted in the signal view widget. Any number of signals can be selected in the signal list widget to be plotted in the signal view widget.

   (b) **Remove signal plot:** This function is exact opposite of 'add signal plot'. If the user un-checks a checkbox in the signal list widget, then that signal's plot is removed from the signal view widget.

4. **'Console' widget:** In this widget, once the pattern recognition is performed, its output is displayed. If there are any malfunctions found during the pattern recognition step, then the malfunction name, instance name, timestamps at which the malfunction occurs and the signal names are displayed. If no malfunction is found, then 'No malfunctions found!' is displayed.

5. **'Add new error' button:** This button gets activated only when at least one signal is selected in the Signal List widget. All the signals which are selected in the signal list view can then be added to the database as either a new malfunction or as another instance for an existing malfunction.

In this section we saw in detail how the Automated Trace Analysis tool is developed and how its various components work together. We also saw how the graphic user interface is built.

3.5 Database

This is the final section in this chapter. Here we deal with how the database is structured and what are its functions.

In this thesis, a SQLite database has been used. The various databases available and their merits and demerits have been discussed in the section 2.3. One of the main reasons for choosing this database is because it uses the SQL syntax but as the name suggests, it is a light version. It is easy to implement. Unlike MySQL, this does not require any server set up. Also there is no need for username and password. It is just like any other flat file.

Although SQLite is similar to a flat file, it has its advantages when compared to the latter. The advantages can be listed below:

1. In SQLite, your data can be structured as a database. Which means that it can be easily queried. So we get the functionality just like a normal database; adding new data into the database and retrieving it later.

2. In SQLite you can have multiple tables in one database file. In flat file format you would need a separate file for each table.

3. In SQLite, you can query just the data which you want. But in a flat file you would need to load the full file before querying the data which you want.

4. When an entry is added or edited in the SQLite database, only that part of the file needs to be saved. But the same scenario in the flat file would require the whole file to be re-saved. This would mean that the performance of the SQLite is better than the flat file.

By seeing the above comparisons you can see why SQLite is the best database for the automated trace analysis tool.

In my SQLite database, I have 3 tables. A class diagram of this is shown in the Fig 3.11. The tables and their descriptions are given below:

1. **Error Table:** This is the topmost table. This table has 2 columns.
   (a) **ID:** This column stores the ID of the malfunction.
   (b) **Error Name:** This column stores the malfunction name which the user inputs.

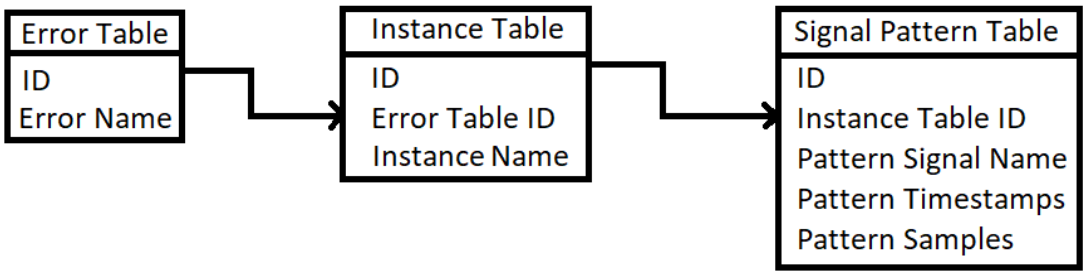

Figure 3.11: Database Tables

2. **Instance Table:** This table acts as a conjunction between the first and the last table. This table has 3 columns.

   (a) **ID:** This column is used to store the ID of the instance.

   (b) **Error Table ID:** This column stores the ID of the error table which connects to this instance.

   (c) **Instance Name:** The user inputted instance name is stored in this column.

   A single malfunction can have multiple instances. Hence multiple rows in this table can have the same Error Table ID.

3. **Signal Pattern Table:** This is the lower-most level of the database. In this table the details of the actual signals which are involved in the pattern recognition are stored. This table has 5 columns.

   (a) **ID:** This column is used to store the ID of the signal.

   (b) **Instance Table ID:** This column links the Signal Pattern Table to the Instance Table. In this column the ID of the Instance Table can be found.

   (c) **Pattern Signal Name:** This column is used to store the name of the signal which is used in the pattern recognition.

   (d) **Pattern Timestamps:** The timestamps of the error signal is stored in this column. The array of timestamps is stored in the string format.

   (e) **Pattern Samples:** The sample values of the error signal is stored in this column. Again even in this column, the array of values is stored as a string.

Just like how a single malfunction can have multiple instances, similarly a single instance can have multiple signals which are required for pattern recognition. Hence multiples IDs in this table can have same Instance Table ID.

The database performs 2 operation, read and write. These operations are coded in Python. In Python, the code doesn't quite follow the class diagram given in Fig 3.11. The way it is coded can be represented as shown in the Fig 3.12.

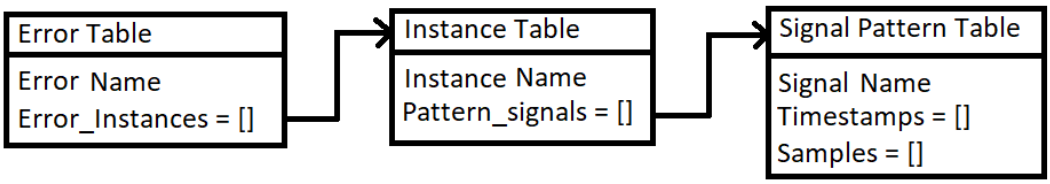

Figure 3.12: Database Table in Python

As you can see from the Fig 3.12, there is no field called ID in any of the tables. Moreover no field called Error Table ID in Instance Table and no field called Instance Table ID in Signal Pattern Table. All the columns which were initially used to connect the current table to its higher table have been replaced with lists.

The Error Table contains a list of instances. The Instance Table in turn contains a list of all the pattern signals. The timestamps and samples are also stored as lists in the Signal Pattern Table.

The database performs 2 functions. How they perform it and pseudo code for it is given in the following subsections.

3.5.1 Reading from the Database

In this section we see how data is read from the database.

When the user uploads a trace file, the first thing which is performed is pattern recognition. To perform pattern recognition, data present in the database needs to be read and given to the different pattern recognition algorithms. The way the read function is performed is as follows:

1. All the rows in the Error table are fetched.

2. For each row in Error Table, all the rows in the instance table with Error Table ID equal to ID of the row in Error Table, fetch the instance table rows.

3. For each row in Instance Table which was fetched, fetch the rows of the Signal Pattern Table with Instance Table ID which matches the ID of the row in Instance Table.

4. For all the rows fetched from Signal Pattern Table, get their signal name, timestamps and samples.

The pseudo code of this is simple and is given in the listing 3.7.

```
# Fetch all the rows from Error Table

for error_table_row in Error_Table:
    cursor.execute('SELECT * FROM Error_Instance_Table WHERE
    Error_Table_ID=?', (error_table_row[0]))

    # Fetch all the rows from Instance table which satisfy the
    above condition.

    for error_instance_table_row in Instance_Table:

        self.cursor.execute('SELECT * FROM Pattern_Signal_Table
    WHERE Error_Instance_Table_ID=?', (error_instance_table_row[0]))

        for error_instance_pattern_signal_table_row in
    Pattern_Signal_Table:

            # Fetch the signal name, timestamps and samples from
    Pattern Signal Table of all the rows which satisfy the above
    condition .
```

Listing 3.7: Pseudo Code for Reading the Database

In this subsection we saw the read function which is present in the database. The coding of this function is done in Python language.

3.5.2 Writing to the Database

This is another functionality which is performed by the database.

Data is written into the database when the user selects the button 'Add New Error' which is present in the GUI of the tool. This button can be seen in the Fig

3.10. When the user presses this button, he is prompted to a new window which is shown in the Fig 3.13.

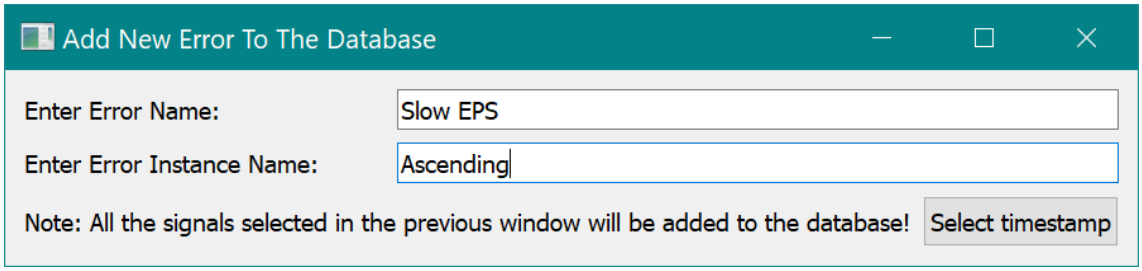

Figure 3.13: New Window to add the Malfunction to the Database

In this window the user needs to provide a name for the malfunction and also a name for the instance. Once both these details are provided, the 'Select timestamp' button gets activated. The user is also notified that the signals which he has selected in the signal list view widget are the ones which will be stored. So if the user wants to save patterns from different signals, he can just close this window and select the signals again. Once the user presses the 'Select timestamp' button, he'll again be prompted to a new window with the plots of the selected signal. This is shown in the Fig 3.14.

As we can see in this Fig 3.14, the user can freely select the timestamps in each signal. Once he has selected the timestamps in all the signals, the button 'Add' gets activated. This button, when pressed, adds the malfunction to the database along with the selected timestamps and samples.

The algorithm of writing data into the database can be explained as follows:

1. Fetch the error name and instance name which the user inputted.
2. If the error name already exists, then make a new entry only in the Instance table with the instance name and the existing Error Table ID.
3. If the error name doesn't exist, then make a new entry in the Error Table with the given error name. Also make a new entry in the Instance Table with the given instance name.
4. Go to the Pattern Signal Table and add all the signal names, their corresponding timestamps and samples in the string format.

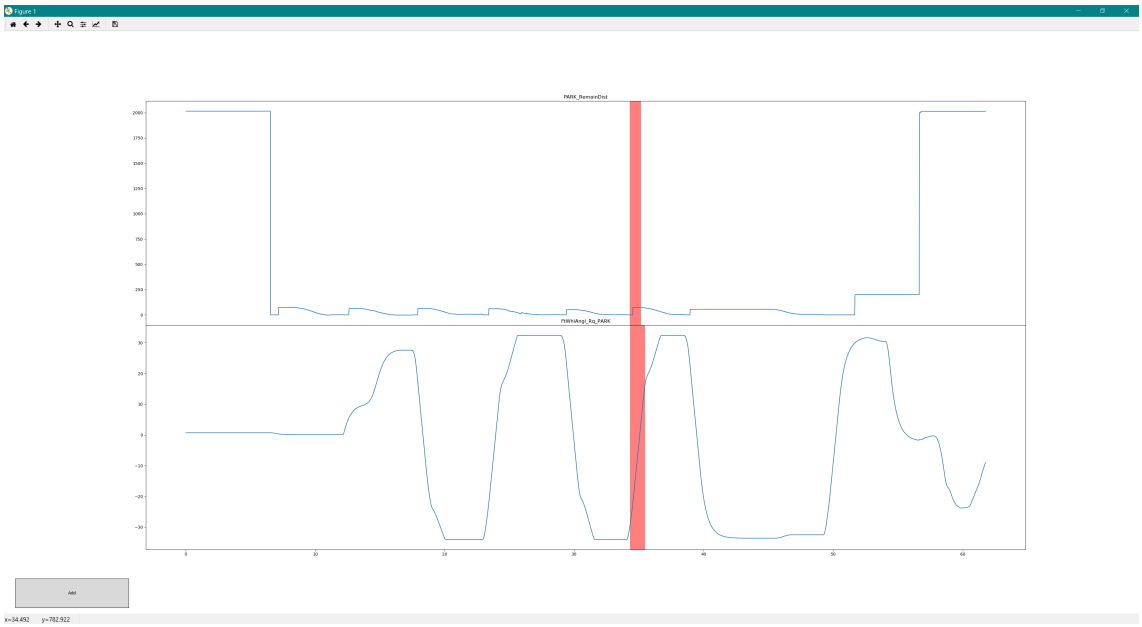

Figure 3.14: New window to select the timestamps

A pseudo code of this is given in the listing 3.8.

```
# Check if error name already exists in the database
if(Error_Name exists in Error_Table):
    # Fetch the ID of the existing Error_Name

    cursor.execute('INSERT INTO Instance_Table(Error_Table_ID,
    Instance_Name) VALUES (?, ?)', (Error_Table_ID, Instance_Name))

    for pattern_signal present in pattern_signal_selected:
        # Store the pattern_signal_name, pattern_signal_timestamps
    and pattern_signal_samples

else:
    cursor.execute('INSERT INTO Error_Table(Error_Name) VALUES (?)'
    , (Error_Name))

    cursor.execute('SELECT MAX(ID) FROM Error_Table')

    # Get the maximum ID in the Error_Table

    cursor.execute('INSERT INTO Instance_Table(Error_Table_ID,
    Instance_Name) VALUES (?, ?)', (Maximum_ID, Instance_Name))
```

```
18
19      for pattern_signal present in pattern_signal_selected:
20          # Store the pattern_signal_name, pattern_signal_timestamps
    and pattern_signal_samples
```

Listing 3.8: Pseudo Code for Writing to the Database

In this subsection we saw how to write to the database. Coding of this has been achieved using Python language.

In this whole section, we got an overview of the SQLite database and its versatility. We went into detail of the different functions performed by the database.

We now come to the end of this chapter. We saw the different ways to detect a malfunction. The detailed description of the automated trace analysis tool. The evaluation of all these will be seen in the next chapter.

4.Evaluation

In the previous chapter we saw the two different approaches which were implemented as a part of this thesis. Now we shall evaluate each method and see how good they are.

4.1 Evaluation of 'Hard-coded' Approach

In this approach the values for detecting the malfunctions were hard-coded.

The merits of this approach is very limited. The merits of this method are given below:

1. It is easy to code.
2. The implementation is simpler.

But the demerits of this approach exceeds its merits. Demerits are as follows:

1. Implementation is based on a per malfunction basis.
2. Latency affects the detection of defects.
3. There is very little flexibility in coding.
4. The length of the code increases exponentially.
5. Detection of jitter is not possible.

4.2 Evaluation of 'Pattern Recognition' Approach

The detection of malfunctions in this approach is based on the pattern matching.

The advantages of this approach is as follows:

1. This approach is a general purpose detection.
2. The detection of jitter is possible here.

3. One of the biggest advantages of this approach is that the user can manually define a pattern which needs to be detected.

The disadvantages of this approach is as follows:

1. The implementation complexity is higher when compared to the hard-coded approach.

2. It requires a shared database. And only one user can make changes to this database at a time.

From the above mentioned merits and demerits we can come to a conclusion that the Pattern Recognition approach is much more suitable for this thesis. Also it is user friendly. The user doesn't need to know how to code to add a new malfunction to the database.

As of now around 5 people are performing the manual detection of malfunction. The introduction of this tool should help in reducing that number. It is expected that the number will be dropped to only one or maximum two. This would in turn result in more highly-competent people being able to concentrate on the development of other tools.

The database which is used is SQLite. The reasons we to why we use this has already been discussed in sections 2.3 and 3.5. The SQLite even satisfies all the questions which were asked in the section 1.2.

If there ever comes a need for more than one person to simultaneously make changes to the database, it won't be possible with this database.

This is the only drawback of SQLite database. But as described above, this tool is expected to be used to reduce the number of people working on detection of malfunctions. Therefore the chances of multiple people using this tool simultaneously is very low.

5.Conclusion and Future Aspects

This is the final chapter of this thesis. Here we see an overview of all that is done till now and the where all there is a chance for improvement.

5.1 Conclusion

The work done in this can be divided into two parts.

1. **Hard-coded approach:** A brief overview of this approach is as follows:
   (a) This was coded in CANoe tool by Vector Informatik GmBh. The coding language used was CAPL. A screenshot of CANoe is shown in Fig 5.1
   (b) This approach was implemented for 2 malfunctions.
2. **Pattern recognition approach:** The majority of the thesis work was done in this part. This has a lot of processes involved. The following are the work done in this thesis:
   (a) For this approach to be used, a tool called 'Trace Analysis Tool' has been developed. GUI implementation for the tool is done in Python. A screenshot of this tool can be seen in Fig 3.10.
   (b) Code for conversion of BLF trace file to MDF trace file is performed using C# code.
   (c) The signal names can be loaded and plotted in the tool.
   (d) 2 types of algorithm for pattern recognition, Discrete signal pattern recognition and Continuous signal pattern recognition, have been invented. The coding of this is done in Python.
   (e) A SQLite database has been created for the tool. Reading and writing operations can be performed on this database from the tool.

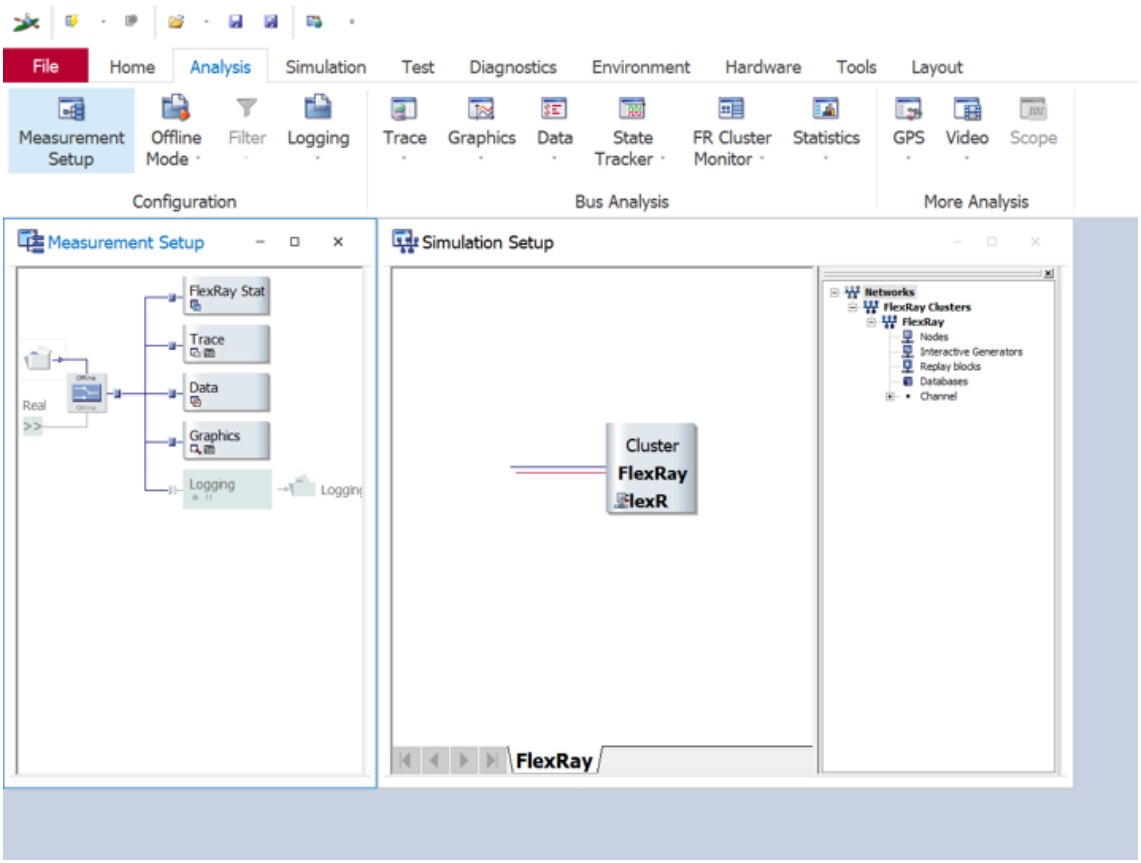

Figure 5.1: A screenshot of the CANoe GUI

5.2 Future Aspects

The Trace Analysis Tool which was developed as a part of this thesis has a lot of potential for improvement. A few of the improvements which can be made are as follows:

1. This tool can be extended to be used by other departments of Daimler AG. This can be done my adding a small feature in the menu bar where the user can upload the file path of the pattern recognition database file. As of now the database file path is hard-coded in the code.

2. Improvisation of the pattern recognition algorithms by fine tuning the scaling factors.

3. Having an option to select an existing malfunction in the 'Add to database' window. This would help the user to know if a malfunction already exists. Then the user can just give an instance name.

4. Can introduce a delete function for removing a malfunction or an instance of a malfunction from the database. As of now if the user wants to delete something from the database, then he has to manually do in from the database. It is not possible to do it from the tool.

5. If the tool is going to be used in large-scale, i.e. multiple people using it simultaneously, then there is a need to change the database from SQLite to a different one.

Bibliography

[1] Tommaso Pifferi, "How to efficiently store and query time-series data." https://medium.com/@neslinesli93/how-to-efficiently-store-and-query-time-series-data-90313ff0ec20. Published on 3 August 2018.

[2] Vector Informatik GmBH, "Logging formats." https://kb.vector.com/entry/520/. Last updated 20 May 2019.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2009.

[5] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3rd ed., 2011.

[6] Jason Brownlee, "Supervised and unsupervised machine learning algorithms." https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. Published on 16 March 2016, Last Updated on 12 August 2019.

[7] J. Lin, S. Williamson, K. Borne, and D. DeBarr, "Pattern recognition in time series," *Advances in Machine Learning and Data Mining for Astronomy*, vol. 1, pp. 617–645, 03 2012.

[8] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, pp. 561–580, Oct. 2007.

[9] Robert Polding, "Databases: Evolution and change." https://medium.com/@rpolding/databases-evolution-and-change-29b8abe9df3e.

[10] Ostezer and Mark Drake, "Sqlite vs mysql vs postgresql: A comparison of relational database management systems." https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. Updated on 19 March 2019.

[11] PostgreSQL, "What is postgresql?." https://www.postgresql.org/about/.

[12] SQLite, "Sqlite is serverless." https://www.sqlite.org/serverless.html.

[13] "Inserting into a database with sqlite." https://pythonprogramming.net/sql-database-python-part-1-inserting-database/.

[14] Tendero, Stack Exchange, "Finding a signal inside another signal." https://dsp.stackexchange.com/questions/51631/finding-a-signal-inside-another-signal.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 25/10/2019

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 25/10/2019