Log data filtering in embedded sensor devices

JAKOB OLSSON, VIKTOR YBERG
Abstract

Data filtering is the disposal of unnecessary data in a data set, to save resources such as server capacity and bandwidth. The method is used to reduce the amount of stored data and thereby prevent valuable resources from processing insignificant information.

The purpose of this thesis is to find algorithms for data filtering and to find out which algorithm gives the best effect in embedded devices with resource limitations. This means that the algorithm needs to be resource efficient in terms of memory usage and performance, while saving enough data points to avoid modification or loss of information. After an algorithm has been found it will also be implemented to fit the Exqbe system.

The study has been done by researching previously done studies in line simplification algorithms and their applications. A comparison between several well-known and studied algorithms has been done to find which suits this thesis problem best.

The comparison between the different line simplification algorithms resulted in an implementation of an extended version of the Ramer-Douglas-Peucker algorithm. The algorithm has been optimized and a new filter has been implemented in addition to the algorithm.

Keywords

Data filtering, line simplification, Ramer-Douglas-Peucker, memory efficiency.
**Sammanfattning**

Datafiltrering är att ta bort onödig data i en datamängd, för att spara resurser såsom serverkapacitet och bandbredd. Metoden används för att minska mängden lagrad data och därmed förhindra att värdefulla resurser används för att bearbeta obetydlig information.

Syftet med denna tes är att hitta algoritmer för datafiltrering och att undersöka vilken algoritm som ger bäst resultat i inbyggda system med resursbegränsningar. Det innebär att algoritmen bör vara resurseffektiv vad gäller minnesanvändning och prestanda, men spara tillräckligt många datapunkter för att inte modifiera eller förlora information. Efter att en algoritm har hittats kommer den även att implementeras för att passa Exqbe-systemet.

Studien är genomförd genom att studera tidigare gjorda studier om datafiltreringsalgoritmernas och dess applikationer. Jämförelser mellan flera välkända algoritmer har utförts för att hitta vilken som passar denna tes bäst.

Jämförelsen mellan de olika filtreringsalgoritmerna resulterade i en implementation av en utökad version av Ramer-Douglas-Peucker-algoritmen. Algoritmen har optimerats och ett nytt filter har implementerats utöver algoritmen.

**Nyckelord**

*Datafiltrering, Ramer-Douglas-Peucker, minneseffektivitet.*
# Table of Contents

1 Introduction .................................................................................................................. 1
   1.1 Background ................................................................................................................. 1
      1.1.1 Line simplification ............................................................................................. 1
   1.2 Problem ..................................................................................................................... 2
   1.3 Purpose ..................................................................................................................... 3
   1.4 Goal ........................................................................................................................... 4
      1.4.1 Benefits, Ethics and Sustainability ......................................................................... 4
   1.5 Methodology ............................................................................................................. 5
   1.6 Delimitations ............................................................................................................ 5
   1.7 Outline ....................................................................................................................... 6

2 The Exqbe system .......................................................................................................... 7
   2.1 Line simplification algorithms .................................................................................. 8
      2.1.1 Visvalingam-Whyatt .......................................................................................... 9
      2.1.2 Reumann-Witkam ............................................................................................ 10
      2.1.3 Opheim simplification ...................................................................................... 11
      2.1.4 Ramer-Douglas-Peucker .................................................................................. 13
   2.2 Previous work .......................................................................................................... 15
      2.2.1 Previous filtering algorithm .............................................................................. 16

3 Developing the algorithm ............................................................................................... 18
   3.1 Exqbe research ......................................................................................................... 18
   3.2 Electrocardiography filter ....................................................................................... 18
   3.3 Algorithm implementation ....................................................................................... 19
      3.3.1 Determining tolerance level ............................................................................... 20
      3.3.2 Buffer implementation ...................................................................................... 22
      3.3.3 First buffer prototype ...................................................................................... 22
1 Introduction

In an age of information we want to store as much information as possible and have it accessible at all time. This is however not sustainable as it would require unreasonably large servers to store the information. A solution to this problem is to save all the information of importance and discard the rest, this is why a suitable and well-designed filtering algorithm is important. A proper filtering algorithm will find important information and save it, e.g. the shape of a graph, an anomaly in an engine or the like.

1.1 Background

Data filtering is used to dispose of unnecessary data, while still keeping enough data to fulfill its purpose of displaying accurate information (1). An information filtering system plays an important part in systems with limitations in storage capacity, bandwidth or performance. Without any type of filtering, servers could get flooded with information and valuable resources are used to process insignificant information.

1.1.1 Line simplification

An approach to data filtering in graphs is to use a line simplification algorithm. Using line simplification will result in data compression and reduction in the amount of data points by removing redundant points
and thereby saving storage space and transmission cost. Different line
simplification algorithms take different properties into consideration to
remove redundant points from the graph, while still maintaining the
shape of it. Some of these properties are radial distance, which is the
raw distance between two points, and perpendicular distance, which is
the distance between a point and a line formed by two other points and
used for measuring the point's deviation. The key to an efficient data
compression is to find the correct properties and tolerance level (2).

1.2 Problem

A problem with filtering algorithms is to find one that suits the purpose
well. There are many to choose from and they all excel in different areas.
To find a suitable filtering algorithm there are some factors to take into
consideration:

- What kind of data needs filtering?

In this project the data that needs filtering is mainly focused on
Revolutions Per Minute, RPM, in engines but data such as voltage, oil
pressure and other sensor values can also be looked at.

- What kind of filtering is needed? (save a curve, save anomalies,
  save constant values)

STS are looking for a filtering algorithm to accurately maintain the
shape of a graph. For other systems filtering things such as anomalies,
separating segments of graphs or filtering land border graphs may be of
interest.
• How precise does the filtering need to be?

A filter with the possibility of varying precision is what is searched for in this project as different sensors may want different precision.

• What are the constraints? (local memory, storage capacity, performance)

For the central device there are several constraints to take into consideration, mainly bandwidth and server storage are the concerns, but the algorithm has to be developed to fit a system with local memory constraints as well.

This thesis will examine what approaches there are to achieve data filtering with good memory efficiency while maintaining the shape of the graph.

1.3 Purpose

When logging information from a continuous stream of data, systems may get too much information to handle which will cause a waste of power, performance and money. A solution to this problem is to not store all information, but to separate what is important from what isn’t.

The purpose of this bachelor thesis is to find and implement a filtering algorithm that removes unnecessary data points from a graph, and create necessary extensions and modifications to achieve said filtering. It will also describe the process of developing a prototype of the filtering algorithm.
1.4 Goal

By using a filtering algorithm data storage service providers and network service providers will benefit by saving server capacity and bandwidth, due to no redundant data points being stored and transmitted.

The result of the project will be an algorithm which success can be measured by seeing how well it executes compared to the previously used algorithm and in an overall reduction of data points.

1.4.1 Benefits, Ethics and Sustainability

Sustainability in this project would be how resources are being used or how it affects the amount of resources being used. With less data to process, less resources are being used by both the device and the server. The server will be able to hold more significant information without being flooded. This will lead to a more sustainable server infrastructure as there is less data to store per client, which results in one server being able to serve more clients. This also affects economic sustainability as less resources being used means less cost.

As for the social sustainability which can be considered how this project affects working environments and livability in a social environment, it doesn't have any affect other than the indirect ones which could come from less cost and better economic sustainability.
1.5 Methodology

During the project there was close interaction with the stakeholder, and an agile methodology would be most suited for the project. Two known agile methods are Scrum and Dynamic Systems Development Method (DSDM). Scrum is a way to distribute work assignments in a group. In Scrum different roles are assigned and specific requirements from the stakeholder are written down and broken down into sprint planning (3). DSDM is a project management method where prototyping is seen as more important than detailed planning (4).

The theoretic research was performed with an inductive approach. With this approach, knowledge is gained by observation and reading, as opposed to a deductive approach, where theses are formed from theories and experience. The inductive approach is often used to study an area in which the researcher lacks expertise. Research thesis and questions can be used to narrow the scope of the study (5). Based on these facts an inductive approach was most suited.

1.6 Delimitations

To delimit the problem, this study focuses on finding the best algorithm to filter data with respect to memory efficiency and bandwidth usage. A device with limited memory capacity requires a memory efficient algorithm. To reduce bandwidth cost, the filtering should also minimize information being sent to the cloud.

Things that could affect the result but is not taken into consideration for this study is sensor margin of error. A sensor could track incorrect data
and throw the algorithm off and giving incorrect data to compress, resulting in a faulty graph. Since the algorithm executes in such an early stage of the data handling, only tracking errors can occur.

1.7 Outline

The report is divided into six chapters and should be read in a chronological order.

Chapter two explains the theoretic background such as specifics on the Exqbe system, line simplification algorithms and previous work.

Chapter three describes the work process and how the product was developed from start to finish.

Chapter four show the results from the finished algorithm, primarily showing statistics and comparison by using test values.

Chapter five shows our own thoughts about the process, an evaluation of the algorithm and potential future work.

Chapter seven in appendix describes how the project has been developed, i.e. what methods have been used to develop the project and how the literature study was done.
2 The Exqbe system

Exqbe is a system by Scandinavian Technical Solutions (STS) developed for monitoring vehicles. The system was originally developed to monitor vehicles without this type of functioning built-in, e.g. older cars or boats. It consists of a central device, some sensors and a software infrastructure around it, such as cloud storage and an app to display real-time information from the system. The system can be installed in a car to monitor information such as battery voltage, movement, position and temperatures. A total of 16 sensors can be connected to the central device using hubs, an extension for distributing functionality and minimize cabling (6).

The central device is equipped with a buffer in which information is stored before it is sent off to the server. If there is no internet connection, there is a flash memory installed that can store data locally.
before sending it to the cloud server once connection is reestablished. The device transmits data to the cloud over 3g and uses Bluetooth to communicate in real-time with a smartphone or a tablet. Data sent to the cloud uses the HTTP protocol (7). It can also be equipped with an alarm system that monitors movement and position when the car is parked. If the alarm is triggered, the owner gets notified via e-mail or SMS (8).

Sensors read analog or digital signals from objects in the vehicle. An analog signal is measured as a resistance and converted to a digital signal with an analog-to-digital converter, using a microcontroller. The signal is then sent to the central device where the data is being filtered. The central device uses a 32-bit microcontroller with 512 kB of RAM, where the memory is used for both the software and local storage. The local storage uses a buffer with around 700 bytes of memory (7).

The sensors are tracking a possible infinite stream of data that needs continuous filtering. The filtering algorithm can therefore only filter incoming data with respect to previous data, not one session as a whole, and therefore it needs some adjustments to fit the system.

2.1 Line simplification algorithms

A line simplification algorithm is an algorithm that traverses a graph and removes unnecessary data points. It can be used both to render high-resolution polylines such as land borders and coast lines to make them more visually clear or to save storage space by reducing the amount of data (9).
There are some different approaches to graph filtering. In general, an algorithm will create a line based on two key points and then measure the perpendicular distance (shown in figure 2.2) or radial distance (shown in figure 2.3) between the line and a selected test point.

An incremental algorithm will look at the points before the selected test point and decide whether to keep it or to remove it. A holistic algorithm will analyze the graph as a whole and remove unnecessary points not only depending on previous points.

![Figure 2.2 Perpendicular distance. The distance to the green point is larger than the tolerance level and will be saved.](image)

![Figure 2.3 Radial distance. The red point is within the maximum radial distance of the first point and is removed.](image)

### 2.1.1 Visvalingam-Whyatt

The Visvalingam-Whyatt line simplification algorithm was developed in 1993 by M. Visvalingam and J.D. Whyatt, and is also known as *Line generalization by repeated elimination of points*. The algorithm will value each point in a graph based on its significance and place the point on a stack, meaning the most significant points will be on top of the stack, and the least on the bottom. A significance value is given based on its effective area, meaning the change in total area if this point is
removed. The algorithm will then take \( n \) points from the top of the stack and place them back in the graph, meaning the \( n \) most significant nodes will be placed back in the graph (9) (10).

The Visvalingam-Whyatt algorithm uses a stack that will put additional strain on memory usage which is something that in this project should be kept to a minimum due to the limited local memory available. Another issue is that the algorithm can only work with a few values at once and therefore it may become a problem because signing a significance level to e.g. ten values and choosing the five most significant doesn't mean these five are actually significant in a bigger graph.

2.1.2 Reumann-Witkam

The Reumann-Witkam algorithm is a perpendicular distance based algorithm. In this method the algorithm will draw a line from a key point to the next point and use a strip with the width of twice the tolerance level. If any of the following points are within the tolerance level they will be removed, except for the last point inside the strip that will be used as a starting point (key point) for the next section. This is repeated until the entire graph has been traversed, and the key point is the last point in the graph. This algorithm is incremental, which means that it will only go through the graph one time, making it a fast algorithm (11).
Reumann-Witkam provides good time complexity and uses no additional data structures that can put strain on the local memory, but it doesn't provide the possibility of a high accuracy that is searched for in this project due to it being incremental and therefore this algorithm is discarded.

### 2.1.3 Opheim simplification

The Opheim algorithm is an algorithm based on perpendicular and radial distance. The algorithm will base a radial distance at a key point, a line will be drawn to the last point within the radial distance. If there are no points within the radial distance it will draw a line to the next point. It will then cast a ray, with the width of twice the tolerance level, in that direction and remove points with a perpendicular distance less than the tolerance. If there are no points within the radial distance and
the ray, no points will be removed. If there are several points within, the last point will be stored and the algorithm will repeat with the last point as a new key point (9). Similar to Reumann-Witkam this algorithm only goes through the entire graph once, making it a fast algorithm.

Figure 2.6 A point is within the radial distance and the ray is cast in that direction. The second, third and fourth point are all within the tolerance level so only the fourth point remains.

Figure 2.7 No point is within the radial distance so the ray is cast in the direction of the next point. Both the second and the third point are within the tolerance level so only the third point remains.

Because of the fact that this algorithm filters based on both radial and perpendicular distance means it will filter strict, meaning it will remove a lot of points. For this reason it was decided to go with another algorithm instead as the possibility for a very accurate filtering is necessary.
2.1.4 Ramer-Douglas-Peucker

Ramer-Douglas-Peucker (RDP) is a recursive algorithm that is perpendicular distance based and holistic. The algorithm uses a given tolerance level $\varepsilon$.

- The algorithm will select the first point and the last point as key points, and draw a line between these.

- For each point between the chosen key points it calculates the perpendicular distance $\Delta$ and selects the point with the largest distance as a test point.

- If a test point with a perpendicular distance larger than the tolerance level is found ($\max(\Delta) > \varepsilon$) the algorithm will subdivide by the point with the largest perpendicular distance into two lines and the point is saved. One line between the first key point to the test point, and another line from the test point to the second key point.
• If no perpendicular distance $\Delta$ is greater than the tolerance level $\varepsilon$ ($\max(\Delta) < \varepsilon$), the algorithm will remove all points between the chosen key points.

(12)

• Now the algorithm will recursively repeat these steps with the two new lines until the entire graph has been traversed (2).

(12)

When the entire graph has been traversed with recursion all retained points now make up a compressed graph with redundant points removed (13).
While this algorithm has the worst time complexity out of the mentioned algorithms it is still good enough for this project. The algorithm also provides the possibility of high accuracy and was therefore chosen as the algorithm to work with.

### 2.2 Previous work

A similar study on filtering a continuous stream of data has been done by researchers at TU Eindhoven and Sharif University of Technology (14). This study tracks migratory pattern of animals, where their position is constantly streamed from a GPS tracking device. To reduce data storage data filtering was necessary. To solve the simplification problem they use *Hausdorff distance* and *Fréchet distance*.

*Hausdorff distance* will have set of points $A$ and $B$ and find the shortest distances from all points $a \in A$ to any $b \in B$ and store them in $C$. Then it determines $\Delta$ based on the longest distance in $C$, and retains all points with a greater distance than $\Delta$ (14) (15).

*Fréchet distance* is two points on two curves that are the furthest length from each other. Imagine a dog and its owner walking on separate curves independently. The *Fréchet distance* is defined as the minimum leash required for the owner and the dog to be at any given point, respectively (14) (16).

In the conducted study the filtering is not performed in the transmitting device, and therefore there are no major constraints on the memory usage during the filtering. The only constraint is the memory on the servers storing the data. Therefore, the solutions did not take the limited
availability on local memory into consideration and was not the most suited for this thesis.

### 2.2.1 Previous filtering algorithm

The filtering algorithm previously used by STS was simple. Every new value tracked would be compared to the previously logged value. If the newly tracked value was within an offset limit, it would be discarded as the change was considered too small and unnecessary to store. This method gives varied results, but in general too many irrelevant points are stored and important points risk being discarded as every value is filtered under the same circumstance.

Another problem that arises when using this algorithm is that in every constant segment of the graph, all the points proceeding the last stored value are discarded when a value outside of the offset is tracked. This can result in a faulty deceleration of a graph. All the points in the segment are compared to the previously stored value and if it is not outside of the offset it is discarded. But if the following point is outside of the offset the previous point is significant but has now already been discarded.
Figure 2.8 The blue line represents the filtered graph. The red dot is discarded due to being within the offset of the previous stored point. The black dotted line represents the accurate graph.

This algorithm does not place any weight on different scenarios and compares all points under the same circumstances. However this algorithm can be useful if a constant segment of the graph can be identified and a specific offset is set for each sensor. It is then effective at sorting out values in the constant segment that are unnecessary, but modification would be needed to make sure that no important graph points are lost. This filter works for horizontal straight lines and does not affect straight lines in any other direction.
3 Developing the algorithm

This chapter will describe the development process of the algorithm.

3.1 Exqbe research

To start off the project, more information about how the Exqbe system worked was necessary such as what kind of data is tracked and how it stores information. To learn more about the system, the people from STS went through the specifics and requirements.

Exqbe tracks data from sensors that the user can plug into their vehicle and then decide which sensors to use depending on their preferences or need to track data. The device will store data locally in a buffer before transmitting it to a server using 3G. The data will be stored in packets and sent with a timestamp. When the packet is received at the server the data points will be given an equally large time frame within the timestamp given. The device runs in C++ and the server runs Python.

3.2 Electrocardiography filter

Initially, an electrocardiography (ECG) filter was suggested to look into. ECG is the process of monitoring electrical activity of the heart using electrodes. The electrodes detect tiny changes in the skin that originate from the heart. This method will record a lot of distortion, making the
graph difficult to read. To remove the distortion, filtering is needed to make the graph readable and usable, this means that an ECG filter will minimize distortion in a graph. After presenting this to STS it was determined that this was not suitable for the system as the goal of the algorithm is to reduce the number of points on a graph rather than making it more readable.

Instead, research in the line simplification field was done. With line simplification data is reduced, rather than just made more visually clear.

### 3.3 Algorithm implementation

A filtering algorithm with the intent of keeping the shape of the graph should be holistic to yield the best result. It needs to take all points into consideration, not only the points in close proximity or previous to a selected test point. With a continuous stream of data a recursive or iterative algorithm is required, since an incremental, non-recursive algorithm will go through the graph only once, not taking later points in consideration.

Based on previous studies, Ramer-Douglas-Peucker proved to yield the best results when working with graphs (9). It is holistic, non-incremental and has the worst-case time complexity $O(n^2)$, and the average time complexity $O(n \log n)$. It doesn’t use additional stacks or other data structures for filtering the graph, making it memory efficient. It is important that the algorithm works in-time, meaning that the processing time per filtering should not take longer than the update frequency of the sensor.
The algorithm is designed to take all available points into consideration, but due to the continuous stream of data, in theory it can only take previous points into consideration, not future points.

To implement the algorithm it was first translated from skeleton code to Java, due personal preference and expertise. After the product was finished it was then easier to translate to C++. In addition to the filtering algorithm, a graph drawer was developed in Java to graphically display the result of the filtering.

### 3.3.1 Determining tolerance level

After the foundation of the algorithm is coded a fitting tolerance level has to be determined. To decide a tolerance level to a sensor the simplest way is to test existing sessions to find a value that gives a good accuracy and saves an appropriate amount of points. The figures below are extracts from a test driving session for 6.1 kilometers during roughly 8 minutes and the table is from the entire session. This session is one among many to decide an appropriate tolerance level, and it shows that 0.5 tolerance level was the highest filtering possible without modifying the shape of the graph.
<table>
<thead>
<tr>
<th>Tolerance level</th>
<th>Points after filtering</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2225</td>
<td>0%</td>
</tr>
<tr>
<td>0.25</td>
<td>1421</td>
<td>36.13%</td>
</tr>
<tr>
<td>0.5</td>
<td>1140</td>
<td>48.76%</td>
</tr>
<tr>
<td>0.75</td>
<td>1021</td>
<td>54.11%</td>
</tr>
<tr>
<td>1</td>
<td>909</td>
<td>59.15%</td>
</tr>
</tbody>
</table>

Figure 3.1 Unfiltered, 0 tolerance level. Figure 3.2 Filtered with 0.5 tolerance level. Still intact. Figure 3.3 Filtered with 0.75 tolerance level. Has lost its shape to some degree.
3.3.2 Buffer implementation

Each value stored in the buffer takes 4 bytes and the buffer can hold up to 10 values at once, which means that the size of each buffer is 40 bytes for the values alone. Each value has an associated timestamp of 2 to 8 bytes, depending on optimization. The total amount of sensors that can be connected to the central device is 16, which means that there needs to be 16 buffers. Altogether the memory allocated for the buffers is between 672 and 768 bytes depending on memory optimization.

Due to the limited local memory available the algorithm can’t filter the entire graph at once. Therefore, it needs to be modified to filter up to ten values at a time. A buffer has to be implemented with the goal of having the same results as if the entire graph was filtered as a whole.

3.3.3 First buffer prototype

The first approach was to fill the buffer and then filter the values. The points left after the filtering would stay in the buffer and the sensors would fill the buffer with new values. The new values gets filtered together with two of the old values, acting as reference values. The algorithm will keep filling and filtering the buffer to the point where the buffer is full after filtering, meaning no values were discarded during the latest filtering. Then it transmits the values to the server, empties the buffer and the process starts over. The problem with this approach is that it leaves some unnecessary points between the parts where the buffer is emptied and filled again.
The second approach was to use the “sliding window” technique, used in e.g. the Transmission Control Protocol (TCP) and suggested in a previous study about compression algorithms by Nicola Höhle, et al (17). Similar to the first implementation the algorithm waits for the buffer to be full and then filters it, if the buffer is still full this technique would instead of emptying the entire buffer after filtering only send the first value and slide the other values one step to the left to fit one new value. This technique allows processing of new values based on a partial overlap with the previous values (18). It also solves the problem with data only being filtered with previous data. Initially, new values are filtered with old ones and after some time, it will be the oldest value in the buffer and get filtered with newer values.
Figure 3.6 Illustration of how the sliding window technique is used. After filtering the values 2-11, if nothing is discarded and the buffer is still full, the value 2 will be sent to the server and the future value will be added to the buffer.

The sliding window technique will result in more calculations, as the buffer will be filled more frequently, but it will eliminate the problem with unnecessary values at buffer emptying, as the buffer is never completely emptied. It showed that the device was capable of performing the calculations needed without hindering its performance, which made this approach the best. With this technique, the algorithm can filter a segment larger than ten values, even though there are only ten values in the buffer.

Figure 3.7 Unfiltered accelerating graph.  
Figure 3.8 Filtered using the sliding window technique.
Here is a pseudo code example of how the sliding window technique is implemented in this project. In this example, the buffer is a list of integers.

```plaintext
crnt_buffer_size := 0
buffer := []
while connected do
  get input from sensor
  crnt_buffer_size := crnt_buffer_size + 1
  if buffer not full then
    add input to buffer
  if buffer is full then
    filter(buffer)
    if buffer is full then
      transmit(buffer[0])
      for each i from 0 to crnt_buffer_size - 2 do
        buffer[i] := buffer[i+1]
```

The algorithm waits for input from the sensors and adds the values to the buffer. When the buffer is full, the values are filtered and when no more values can be discarded, the oldest value is transmitted to the server and removed from the buffer, the rest of the values are slid one step to the left and a new value is added.

### 3.4 Analysis of implementation so far

After the algorithm had been implemented, STS provided data from a real session to test the algorithm. The algorithm showed good result in keeping the curve of a graph accurate with few data points, but would keep some irrelevant values when the data alternates between values in close proximity, e.g. 2 920 and 2 910. A solution to this was to increase the tolerance level, but that would in a lot of cases ruin other parts of the graph by saving too few data points.
3.5 Modification of algorithm

To solve the problem where too many or too few data points were filtered, two different approaches were tested; a dynamic tolerance level, dependent on the current shape of the graph, or extending the RDP filter by creating a constant value filter.

3.5.1 Dynamic tolerance level

The first idea was to create a dynamic, scaling tolerance level that would be determined by an algorithm analyzing an interval depending on maximum, minimum and average perpendicular distance and the amplitude.

A significant amount of time was spent trying to tweak the values of the dynamic tolerance level but eventually it was decided that an accurate dynamic tolerance value would not be feasible. This because of the fact that a segment of the graph stored in the buffer could be an acceleration

Figure 3.9 Unfiltered acceleration and alternating values.

Figure 3.10 Filtered with the RDP filter using the sliding window technique, alternating values in close proximity are not filtered properly.
and then subside into alternating between two different constants. It is hard to get an overall impression of the graph with such a small segment size.

### 3.5.2 Constant value filter

The Ramer-Douglas-Peucker algorithm cannot, without a dynamic tolerance level, filter alternating values in close proximity since the deviation of some points sometimes becomes too large. Therefore, a separate algorithm is needed to filter those values. A constant value filter that is executed before a value is added to the buffer has to be implemented.

The constant value filter will compare a new value to the previous value stored in the buffer. If the difference is within a set offset range, the new value will not be added to the buffer right away. The algorithm will then skip through all values within the set limit, always saving the previous value. Once a value is recorded that is outside the offset range compared to the last value in the buffer, the previous value and the recently tracked value outside the offset are both placed in the buffer. This means that the first and the last point on the constant line of skipped values will always be saved to make sure that the shape of the graph is intact and no information is lost.

Another thing that has to be decided is what tolerance level to use, to come up with this value some testing is required. When a value is placed in the buffer it has already been rounded, e.g. to closest ten, and after several tests it showed that the best way to use this constant value filter was to set the offset to the round-off value. By using this as the offset
value it will solve the issue of data in a segment alternating between close values which is a common occurrence in some engines.

Here is a pseudo code implementation of the constant value filter.

```plaintext
skipped := 0
custom_value_filter(new_value) =
    if absolute(prev_stored_value - new_value) <= offset then
        prev_checked_value := value
        skipped := skipped + 1
        return -1
    return skipped

while connected do
    get input from sensor
    if constant_value_filter(input) > 0 then
        add prev_checked_value to buffer
        add input to buffer
        prev_stored_value := input
        skipped := 0
    if constant_value_filter(input) = 0 then
        add input to buffer
        prev_stored_value := input
```

Figure 3.11 Unfiltered acceleration and alternating values.

Figure 3.12 Filtered with the RDP filter using the sliding window technique and the additional constant value filter
The algorithm checks if a new value is within a given offset. If the difference between the previously stored value and the new value is less than the offset, it is not added to the buffer. If the difference is greater than the offset, it is added to the buffer together with the previously checked value to maintain the shape of the graph and not lose significant points.
4 Result

After the algorithm has been developed it is tested using pre-existing measured values. The algorithm is temporarily modified so a string of data could be entered and filtered using the sliding window technique. This was done to give an accurate result as if it is measured in real time and filtered by the algorithm. Testing this way gives as accurate results as possible and the conditions in which the algorithm would be used is simulated.

The testing mainly involves measurements of revolutions per minute (RPM) in a car engine.

To evaluate and present the result of the filtering, a graph drawer developed in Java is used.

4.1 Air purifier

The values used for testing has been measured during roughly a six hour period in which an electrivcal motor in an air purifier has been running. In this measuring period it has tracked data with 1Hz frequency over 21039 seconds. When these values are filtered using the previous STS algorithm it results in 329 points. When the points is filtered with the extended version of the Ramer-Douglas-Peucker algorithm it results in 12 points. This gives a reduction of 96.35% from the previous algorithm,
and a total reduction of 99.94% data points. Due to the motor having a preset constant speed, few points are relevant.

<table>
<thead>
<tr>
<th>Graph shape</th>
<th>Sensor</th>
<th>Raw</th>
<th>STS filter (reduction %)</th>
<th>Extended RDP filter (reduction %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>21 039</td>
<td>329 (98.44%)</td>
<td>12 (99.94%)</td>
<td></td>
</tr>
</tbody>
</table>

**4.2 Car engine**

The algorithm has also been tested using data measured from a BMW 330i driving around a racing track, which causes varying motor speed and a lot of significant data points. The test run took place during about 9 minutes, 530 seconds, and data was tracked with a 5 Hz interval resulting in a total of 2 650 data points logged. The previous filtering algorithm results in 2 174 values post filtering. After filtered by the
extended RDP filter, 428 points remain. This means a reduction of 80.31% compared to the STS filter and 83.85% reduction compared to raw data. Post filtering the shape of the graph still remains intact.

<table>
<thead>
<tr>
<th>Graph shape</th>
<th>Sensor</th>
<th>Raw</th>
<th>STS filter (reduction %)</th>
<th>Normal RDP filter (reduction %)</th>
<th>Extended RDP filter (reduction %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>2 650</td>
<td>2 174 (17.96%)</td>
<td>638 (75.92%)</td>
<td>428 (83.85%)</td>
<td></td>
</tr>
</tbody>
</table>

Another test was performed with a different car driving 6.1 kilometers in Västerås, Sweden, during nearly 8 minutes. The car had an average speed of 47 kilometers per hour. This test could represent a short ride to work.

<table>
<thead>
<tr>
<th>Graph shape</th>
<th>Sensor</th>
<th>Raw</th>
<th>STS filter (reduction %)</th>
<th>Normal RDP filter (reduction %)</th>
<th>Extended RDP filter (reduction %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>2 225</td>
<td>1 765 (20.67%)</td>
<td>1 140 (48.76%)</td>
<td>459 (79.37%)</td>
<td></td>
</tr>
</tbody>
</table>
4.3 Oil pressure

To test the algorithm with other sensor data, measurements of the oil pressure of the BMW was used. Again the test took place during 530 seconds and was tracked with a 5 Hz frequency resulting in a total of 2650 logged points. The STS filter results in 1008 points and the extended RDP filter results in 348 points. This is an improvement of 65.48% compared to the STS filter and total reduction of 86.87% in data points while keeping the graph intact.

<table>
<thead>
<tr>
<th>Graph shape</th>
<th>Sensor</th>
<th>Raw</th>
<th>STS filter (reduction %)</th>
<th>normal RDP filter (reduction %)</th>
<th>Extended RDP filter (reduction %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Oil pressure</td>
<td>2650</td>
<td>1008 (61.96%)</td>
<td>479 (81.92%)</td>
<td>348 (86.87%)</td>
</tr>
</tbody>
</table>

4.4 Small interval test

To see if the algorithm would perform well on a smaller segment, a smaller part of a larger measurement was used for testing. The interval in the test shows an acceleration and a flattening over a few seconds.
The last two tests are measured from a car driving in small residential areas and on the high road for approximately 16 minutes.

<table>
<thead>
<tr>
<th>Graph shape</th>
<th>Sensor</th>
<th>Raw</th>
<th>STS filter (reduction %)</th>
<th>Extended RDP filter (reduction %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>36</td>
<td>30</td>
<td>(16.66%)</td>
<td>8</td>
</tr>
<tr>
<td>RPM</td>
<td>4 965</td>
<td>3 350</td>
<td>(32.53%)</td>
<td>460 (90.74%)</td>
</tr>
<tr>
<td>RPM</td>
<td>5 185</td>
<td>4 542</td>
<td>(12.40%)</td>
<td>595 (88.52%)</td>
</tr>
</tbody>
</table>
4.5 Margin of error

The margin of error in the results are due to some factors that occur before the data reaches the filtering stage. The data filtered in the tests has been rounded to the closest ten, which could alter the output slightly compared to if there were no rounded values, however the difference would be insignificant.
5 Conclusions

The requirements of the algorithm have been met, an efficient line simplification algorithm has been developed and implemented that can be executed on the device's limited memory availability. The algorithm keeps enough data points to keep the shape of the graph intact with a tolerance level that can be adjusted accordingly, depending on what kind of information is being filtered and the needs of a specific usage. This in turn saves bandwidth by having less data packages transmitted from the device to the cloud and server capacity as less storage space is needed.

An unexpected effect of the algorithm is how applicable it is. Initially it was not expected that it could be used on any sensor but rather applied on the ones that were consuming most of both bandwidth and server capacity. But due to how adjustable the algorithm is, it can be applied to any sensor given some adjustments to tolerance and offset. A potential drawback is that a filtering algorithm with better time complexity could have been used, but due to the fact that this algorithm can be used on any given buffer size this is not a problem in this system, as a small one is implemented in the device because of limited local memory. This leads to very few calculations which means that this is not an issue and does not affect the device performance.
It showed that finding an algorithm to use wasn’t necessarily that difficult, there were many researched and developed algorithms already existing, but finding the most appropriate one for this system took some research and understanding of the system. However the most difficult part was implementing the algorithm as several changes and modifications were required to make it viable for the system and the kind of information it was tracking. Changes such as implementing a functioning buffer system and filtering out alternating values proved to be the most time consuming part of the development process.

5.1 Method

The Dynamic System Development Method worked well for us in this project because of the close communication with our stakeholders and due to the fact that the demands put on the algorithm weren’t completely set, but altered during development. Prototype versions of the final algorithm were presented to STS throughout the project to improve towards their needs and expectations. Developing a project like this felt natural to us because of the iterative aspect which constantly strived to develop a better product after presenting the previous one.

If we could change anything with the development process in the project it would be to research an existing and functional buffer technique before we implemented our own. By changing that part of the development we would have saved time and could have worked on fine tuning other parts of the algorithm further.
5.2 Result evaluation

The results show that the extended Ramer-Douglas-Peucker algorithm provides a better filter than the previously used algorithm in all cases, both constant and alternating graphs. It also showed considerably better result than the traditional version of RDP by up to as much as 59.74% (see table 2 under car engine, chapter 5.2). We believe that this extended version can be valuable for all systems that need to reduce storage space and bandwidth. The tests have been performed in both racing environments and normal driving circumstances as well as different kinds of motors.

5.2.1 Tolerance level

The tolerance levels have been picked to filter as much of the data as possible in the tests. If wanted, lower values could be used to filter less values and get more detailed information. If less detailed data is needed, the values could be increased.

Figure 5.1 Low tolerance level (high precision)

Figure 5.2 High tolerance level (low precision)
As seen in figure 6.2, even with low precision filtering that gives a significantly smaller result set, the general shape of the graph remains the same. The left graph has a reduction of 16.28%, from 215 down to 180 points, while the right graph has a reduction of 85.58%, from 215 to 31 points. If users want a high precision filtering, they can use high precision values, to save as many points as possible. If they only want to track anomalies, they can use a low precision filtering to store only the shape of the graph.

The driving tests had short sessions and varying values and still the filtering proved to remove between roughly 80 and 90 percent of the data. In a longer session with a fan motor, 99.94% of points were removed which exceeded our expectations. The reason that such a high percentage of points were removed is because of that very few accelerations and decelerations happening during the session.

We believe that if tests from an average car that drove a longer session was used, the reduction of points would start to trend towards a higher percentage, similar to the one from the fan motor because when driving on a highway for a longer period of time motor speed doesn't give as varying numbers.

5.3 Future work

An improvement to the algorithm could be to implement the dynamic tolerance level so that the user doesn't have to experiment themselves to come up with good preset values. Instead, the algorithm could analyze the input and find appropriate values by itself. This would make the algorithm more user friendly and versatile as the user doesn't have to
have any technical experience or understand how the algorithm works. This would however be difficult because of the fact that the algorithm only works with such a small buffer and accurate values can be hard to establish from that small sample size.

Another feature that can be developed is a risk analysis algorithm, which could be used to predict certain events in the vehicle, i.e. when the tires needs to be filled or when the vehicle needs refueling. The reduced data as a result of the filtering decreases the processing an algorithm would need to do to analyze it.
6 References


7. Karlsson D. Interview. 2015 May 22. dan.karlsson@st solutions.se.


Appendix

7 Development methods

A development method is important to any project to control spending and improve project results. It has shown that using project management methods reduces risks, cuts costs and improves the success rate of a project. Ron Kasabian, the general manager at Intel says that “Good project management discipline stopped us from spending money on projects that fail” (19). Popular software development methods are agile methods, which was used during this project, and the waterfall model.

7.1 Waterfall model

The waterfall model is a non-iterative software development model with focus on detailed planning of the process. The first phase of the waterfall model is requirements. During this phase research are done and outlines for the project are formed. The next step is design, which includes both basic design and technical design such as what modules and programs will be used. After design comes the implementation phase, during this phase source code is written and the product is developed. The next step is verification,
during this step the product is put to test and its functionality is checked. Finally comes maintenance which is ongoing updates of the system (20).

This method was deemed not suitable for the project as no detailed outline of the development plan or product could initially be formed. Therefore agile methods were examined to find if it would be a good fit for the project.

7.2 What is an agile method?

Agile methods are software development processes in which the direction of the project can be decided and altered throughout the process. By having different stages of the project such as sprints or iterations and at the end of each sprint a team must have a product improvement towards completion. Agile methods have focus on repetition or iterations of work cycles in different stages (21).

7.3 Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) is an agile method with focus on prototyping rather than detailed planning of the project. In DSDM a rough outline is first formed according to what the stakeholder requires. After the outline has been formed prototypes can be developed and presented to the stakeholder early, which then allows testing that can be used for improvements to the next iterative phase. In DSDM continuous communication with all stakeholders is required for the work to be efficient (22).
DSDM is a part of the iterative and incremental development model, which is a software development process where the development is done according to the plan-do-check-act, an iterative four-step method including planning, doing, checking and acting. The phases of DSDM are feasibility and foundations, exploration, engineering and deployment. Exploration and engineering are repeated until a desired result is reached (23).

The product was developed using Dynamic Systems Development Method as it was difficult to create a detailed outline of the development process. STS knew what end results they wanted, but not the stages to reach the desired product. Because of this it was required to have continuous communication with stakeholders to ensure that the product met their expectations. Prototypes made it easy to show the progress of the project. Seeing a functioning prototype also made it easier for STS to come up with constructive criticism.

7.4 Literature study

To gain a deeper knowledge in filtering algorithms and their uses, a literature study was performed. The literature study was done by searching for information in books and scientific articles in the Inspec and Scopus databases, Google Scholar and articles online. To find relevant information, the keywords line simplification, filter, algorithm
and *graph* were used in the search queries. These keywords were chosen because they were relevant to the subject and others such as *data reduction* were found by reading the abstract of relevant articles to find more potential keywords. The query was refined until only a few relevant articles remained. The study required information evaluation to find relatively recent articles with credible authors and sources.

In addition to books and articles, interviews were held with Dan Karlsson from STS to find out more about the system.