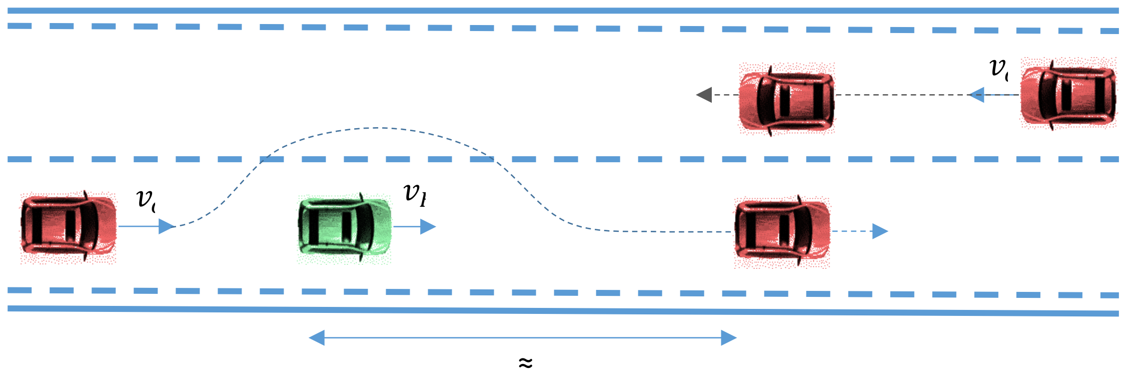
2016-06-07



Automate evaluation of regression test results

Automatisk evaluering av regressions tester

Oscar Martinsson and Fredrik Hansson
Examiner Laura Kovacs

Department of Computer Science and Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2016

Automate evaluation of regression test results

Oscar Martinsson and Fredrik Hansson

Examiner: Laura Kovacs

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
412 96 Gotheburg
Phone : 031-772 1000

Acknowledgements

This thesis work is made by two mechatronics engineering students studying in their last year at Chalmers University of Technology. The education is of 180 points, out of which the thesis work counts 15 points.

The thesis work lasted 10 weeks. We have collaborated with Volvo Car Corporation during those 10 weeks, at the Department of Sensor Fusion. We are very thankful for this opportunity and all support from the object fusion team and our supervisor Robert Mattson at Volvo for this time.

We also want to thank out supervisor Laura Kovacs at Chalmers for the rewarding meetings along the way.

Examiner: Laura Kovacs

Supervisor at Chalmers: Laura Kovacs

Supervisors at Volvo PV: Robert Mattsson

Chalmers University of Technology

Department of Computer Science and Engineering Göteborg, Sweden June 2016

Abstract

Today, Volvo Cars Corporation uses different sensors to get a 360 degree view of how their cars perceive the surroundings and the environment. As part of the car development, here is a need to test and verify the developed functionality.

Currently, SensorFusion department at Volvo Cars Corporation uses automated unit tests (Google Test). However, unit tests are not enough to ensure that the new functionalities do not degrade the product. Regression tests are needed to protect the existing functionality.

At the time of this thesis work, Volvo did not know how to validate results of regression tests in an automated environment where one needs a pass / fail test.

In this report, we develop a test validation algorithm using regression tests. Our algorithm is validated and evaluates output from tested software by comparing it with simulated data. The simulated data used in this thesis comes from the two sensors from a selected traffic scenarios using object fusion output, which is available at Volvo.

In addition to our test validation algorithm, in this thesis we detail our implementation and describe a combination of Matlab files and C ++ code we designed for our work.

Sammanfattning

Idag använder Volvo olika sensorer för att få en 360 graders uppfattning om hur deras bilar uppfattar omgivningen. Som en del av utvecklingen finns det ett behov av att testa och verifiera den utvecklade funktionaliteten.

För närvarande har Volvo Cars Corporation genomfört automatiserade enhetstester (Google Test), men detta är inte tillräckligt för att säkerställa att den nya funktionaliteten inte degraderar produkten. Tester som regressionstester behövs för att skydda funktionaliteten.

Problemet är att Volvo inte vet hur man validera regression testresultaten i en automatiserad miljö där man behöver godkänd/ underkänd test.

I denna rapport kommer vi att utveckla en algoritm av regressions testning för verifiering. Det som kommer verifieras är utdata som jämförs med de simulerade data från två sensorer från ett valt trafikscenarier med utdata från objekt fusion(block som finns på Volvo).

Rapporten behandlar även hur man kan göra om man vill använda Matlab filer i C++ kod.

Contents

1 Introduction

1.1 Background

Volvo Cars Corporation has an ongoing project to design self-driving cars on limited roads in Gothenburg by 2017. Accurate and comprehensive perception of the surrounding environment is paramount for the design of such automated and autonomous vehicles.

Different sensor technologies can be employed for environmental perception systems in vehicles and one sensor unit is unable to monitor the entire environment around the car. Therefore, a perception systems is needed. A perception system consists of multiple sensors covering the 360 degrees area around the vehicle, providing this way a more comprehensive information about the vehicle's surroundings. The combination of the information attained from each sensor, and creating the holistic description of the road, infrastructure and objects around require a process called sensor fusion.

As a part of developing self-driving cars, one needs to ensure reliability of the newly designed functionalities. To this end, testing and verifying the developed functionalities is critical. The Active Safety Analysis and Verification Center at Volvo Cars Corporation is developing a simulation platform where the entire chain from sensor to actuator output can be simulated for different traffic scenarios.

Currently SensorFusion department at Volvo Cars Corporation implemented automated unit tests, Google Tests [1]. Unit tests alone are however not enough to ensure that the new car functionalities do not decrease the overall performance of the sensor module in the car. Several other types of tests are also needed. In particular, regression tests [2] need to be used to protect the new functionalities.

At the time of this thesis, Volvo Cars Corporation did not know how to validate the results of regression tests in an automated environment using pass/fail tests. It was unclear how validation of regression tests could be done without creating "false negatives", and hence creating unnecessary extra work for the organization. At the same time, the organization must also trust the tests not letting any degradation to slip through, which is without creating "false positives".

1.2 Purpose of the Thesis

The present thesis addresses some of the current limitations of the testing technology used at SensorFusion at Volvo. Namely, we develop a test validation algorithm using regression tests. Our algorithm is validated and evaluated by comparing its with simulated data. The simulated data used in this thesis comes from the two sensors from a selected traffic scenarios using object fusion output, which is available at Volvo.

In addition to our test validation algorithm, in this thesis we also detail our implementation and describe a combination of Matlab files and C ++ code we designed for our work.

The thesis has the following two contributions.

1. We are using a simulated scenario where one car overtake another. This scenario is simulated to test two sensors, the 'Side Object Detector' sensor and the 'Radar Camera' sensor. The 'Side Object Detector' sensor is placed at the side of the vehicle and is used to detect objects on the side of the car and have a range of 50 meters. The 'Radar Camera' sensor is placed at the front of the car and is used to detect objects that are about 150 meters ahead of the car.
The simulated test scenarios are formulated as Matlab-files. For setting up our testing environment and using the sensor data from the two different sensors, we present a framework converting Matlab-files into C++ codes. We compare different output from tested code with the ideal simulated data on seven parameters (see Section 1.3); these parameters have been specified by Volvo. We investigate the differences between the sensor data and the optimal simulated data using the Matlab files and evaluate our result.

2. The second contribution of the thesis comes with our regression test algorithm that automatically compares and evaluates the different parameters of the testing environment. The difficult part is to compare the data and see if the compared data is within the range of a pass test, or whether it is a fail test. Our algorithm evaluates whether the tests are better and do not degrade performance since the last test event. If all seven parameters of our testing environment pass the test, then the tested functionality is as good as the last functionality in the exact scenario. Our algorithm is developed with the goal of testing larger code structures before the software tests in the car.

1.3 Delimitations of the Thesis

The work presented in this thesis was carried out within 10 weeks. We have worked full-time on the thesis, eight hours a day. The test should be developed like a unit test structure which make it easy to test in Volvos test environment but when Volvo will use this in there real function they need to move it.

The testing environment of our work is made with two sensors, 'Side Object Detector' and 'Radar Camera' using one usual traffic scenario where one car overtake another. In this test we use seven parameters: speed (in km/h), object, id (of the different object), longitude, latitude, heading and object class/type (ex. car or a truck). Our thesis focused only on these seven parameters and our results have been evaluated using only these parameters. Our algorithm can in principle be used also with other parameters, however, due to the timeframe of our thesis, we validated regression tests using only this seven parameters.

Another limitation of our work comes with its hardware support. In this thesis we used only one computer for working with the thesis code and for designing, implementing and validating the testing process. We had to split up the computer work (implementation, evaluation) among the two of us.

1.4 Research Questions and Objectives of the Thesis

The thesis addresses and answers the following two research questions, corresponding to the objectives of the thesis:

- Which method should be used to evaluate test results and assess of the Sensor Fusion performance pass/ fail test?
- How should we implement our regression tests validation algorithm in C++?

2 Related work

This section overviews methods and approaches relevant to our work.

2.1 Matlab

Matlab stands for a matrix library and is a powerful program which is designed for engineers. It is used in numerous applications from various scientific areas, for example, computer science, robotics, mechatronics, signals and systems, to only name a few. Matlab supports, among others, technical calculations, visualisation of results, method analysis and algorithm developing [1].

Within Matlab, it is possible to export Matlab files to csv text files. In our thesis we used this feature of Matlab to access the test log files in C++.

We used a visualisation program developed in Matlab with different algorithms in order to visualize the 2d model of the vehicle used in our work. We also visualized the different parameters of our testing environment to see and interpret our results, as detailed below.

2.1.1 Export from Matlab to csv-files

To export Matlab files for their use in C++ code, one can follow various paths, depending on various code requirements.

When working with code dependencies, one possibility is to use Matlab coder [2] and export the Matlab files to C++. As a result, one obtains a small Matlab file which start first within the program.

Another way is to use the Matlab library Matio, by using "#include Matio" [3]. As a result, one needs to use the same compiler for Matlab and C++. This turned to be problematic for our work, and therefore we followed another direction, as follows

Targeting an implementation/approach without any code dependencies, one can export Matlab file into regular csv-files [4]. After the generation of these csv files, one can simply use C++ to import the csv text file and use the imported content in C++. This approach has the benefit that it can be used without any code dependencies on Matlab or any other library. This approach is the one we follow in our work.

2.2 Testing of code

2.2.1 Unit tests

During the development of software programs, various functionalities of different modules and program parts need to be tested in order to ensure the reliability of the system [5]. There are many approaches, and accompanying open source programs, that are designed for code testing. One such an approach is called unit testing [1]. Unit testing is an automated approach that invokes units of software programs and checks functionalities about these units. Implementations of unit testing are available and used, for example, in Google Test [11,12], Boost test
and catch [13]. All aforementioned frameworks have the advantage that they can be used to test small building blocks or classes in code before completing the final software product.

C ++ is an object oriented programming language. To develop and test C++ programs, one must often test an entire C++ class or an entire interface. The differences in available testing approaches very much depend on what one wants to develop and validate. For example, how to evaluate the functionality of a method, when one obtains different outcomes from testing the method five times? One can implement five different classes manually or, alternatively, one can use a more general testing approach. The test environment gives the power to define what will happen with the functionality of parts of code classes. One can throw exceptions in the tested code, return different results and make sure functions are handled correct in a specified test suite.

2.2.2 Mocking unit tests

Mocking is a software which is commonly used to complete the test environment and at the same time save both lines of code and testing time [7]. In a unit test, mock objects can be used to simulate the behaviour of complex, concrete objects and hence can be used when it is impossible or impractical to use the concrete object in a unit test. Various mock environments are currently available. In particular, we name here two open-source mock programs, the Turtle Mocks framework and the Google mock framework.

2.2.3 Code review

2.2.3.1 Git

Git is a version handling program designed to ease and speed up tracking and accepting changes in various versions of programs, files and other software objects [9]. This way, Git eases code sharing between software developers and allows changes to the same file in in different versions, so-called branches

In Git, one can create and work with different branches (versions). Updates on one branch are saved locally on one’s computer. This way, Git allows one to make changes to a branch locally, independent to other branches changed by other developers. These changes are then pushed to the main/master branch on GitHub, by merging together the different versions into one. When conflicts between different branches occur, Git provides support to resolve these conflicts and save the changes after resolving conflicts.

In this project we used Git for maintaining our implementation. We cloned our master branch. By cloning the master branch, one gets access to the code resulting from the last update of the master branch and can further continue to develop locally the its own, cloned branch.

2.2.3.2 Gerrit

To get a high code quality on the master branch of Git, one can use Gerrit [10] which is built on top of Git.

When one compares the different versions of code and tracks successful/failing changes between the different versions, one can use Gerrit.

Gerrit is also a web-based code review program that allows one to review other colleagues’ work before they are pushed to the master branch. This turns the master branch the starting point when it comes to code releases.

3 Method

We structured the workflow of our thesis work as follows.

At the beginning of our project, we first a weekly work plan and set the short- and long-term goals for our thesis work.
Next, we proceeded with collecting data and information (Section 3.1), including related work, relevant to our work.

Further, our work focused on the Google test environment and aimed at understanding various design principles of the C++ programing language in which our testing and validation algorithms would be implemented (Section 3.3). To ensure that the test data was well-stored and processed, we translated Matlab codes into csv files and analysed the results of this conversion (Section 3.2).

Finally, we designed our testing algorithm (Section 3.4) and implemented in C++ (Section 3.5). We evaluated and visualized our regression testing algorithm using our new test scenarios (Section 3.6).

3.1 Data Collection

The project required a deeper knowledge on the various programming languages and the applications we used. Therefore we searched for information in different ways, primarily through online search on topics of our project.

3.1.1 Oline seminars/ tutorials

In order to start up and use Volvo's file handling programs, we studied the basics at an online seminar on code version handling system [10]. In order to get ourselves familiar with the coding principles required by the project, we studied various classes/tutorials discussing the difference between C code and C ++.

3.1.2 E-Books/ Code examples

There are many literature studies available comparing programming languages and principles, in order to provide an overview on possible problems and their solutions. One online source with many programming advices and answers to questions is Stackoverflow [6] we used this web site extensively during our project.

3.1.3 Knowledge through lectures/ technical meetings

In the project, we had a number of technical meetings at Volvo. During these technical meeting, we were introduced to the programs used at Volvo and we were shown many code examples, different functionalities and test program tutorials.

3.1.4 Knowledge exchange with colleagues

During the project, we had encountered some difficult problems and questions on formal testing and verification. We have discussed these questions with our qualified colleagues at Chalmers and Volvo. Throughout these discussions, our colleagues also helped us in getting started with the different programs and computer environment. At the beginning, our supervisor introduced us to the Google test environment and we tested together some code and solved some problems in the test environment at Volvo.

3.2 Developing the environment to export Matlab

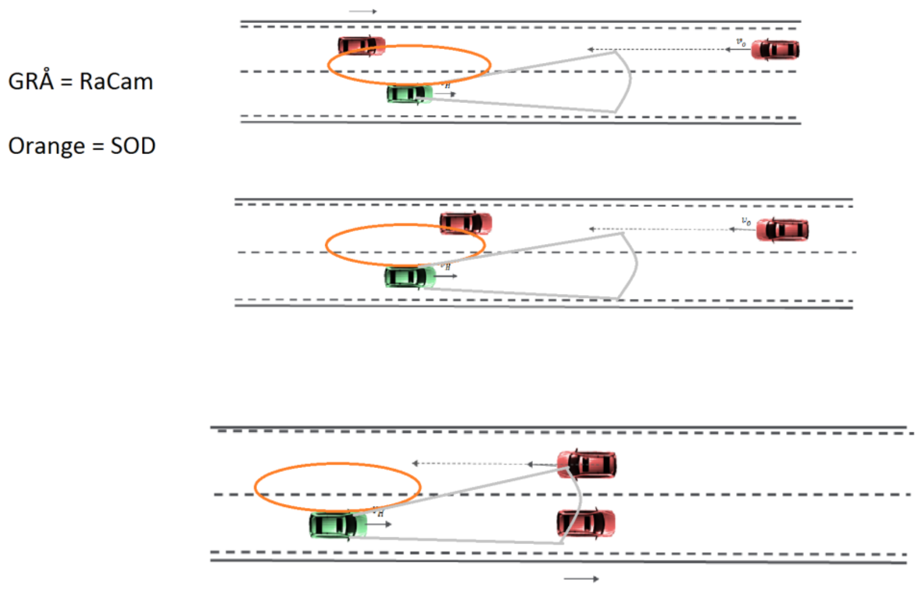


Figure 3:1 Illustrating the testing scenario over time. Orange marking is the SOD sensors and the grey is the Racam sensors.

Our inputs are Matlab file describing the behaviour and corresponding to the car simulations. Our first task to convert these Matlab files to C++ code, as our testing algorithm will be developed in C++. To convert Matlab data into C++ code, we analysed in detail the structure of the Matlab file which contained many use of advanced array structures. After understanding the structure of the Matlab files, we derived the testing scenario to be used further in our regression test validation algorithm. Our testing scenario is illustrated in Figure 3:1 it uses two sensors SideObjectDetector with a range of 50m and a combine Radar Camera sensor with a range at 150 meters. The scenario in figure 3:1 contains cars overtaking each other over time.

3.2.1 Converting the Matlab structure .mat into C++

In our initial attempt, we used Matlab coder to convert the content of Matlab files into a C++ code (see Section 2.1). We also used the Matlab compile feature for converting Matlab files into C++. We however failed for doing so, because of code dependences among Matlab functionalities and files. We next tested the Matio library of Matlab, however we again failed to export Matlab files to C++. We thus came up with a different solution, as follows.

We exported the content of Matlab files to text files of .csv type. These .csv files we could then later read, and hence use, in C++. However, even with this solution we had solve various challenges. When we first exported Matlab files, we could only only export non-negative integers to C++. That is, with a straightforward implementation, we were not yet able to export floats and negative numbers from Matlab to C++. This limitation was critical as we need floats and negative numbers, in addition to integers, to get an accurate cooperation between Matlab and C++ and to capture accurate computations in our testing environment. After investigating possible solutions, we found a way to solve this limitation by using the Dlmright command of Matlab and the 'precision','%.4f':

Dlmwrite (SfOutput_savePath,[SfOutput_part1,SfOutput_part2],**'precision','%.4f'**);
This way, we could export numbers with/without exponents and decimals.

SfOutput_savePath, = The place where the output reference file is created.
SfOutput_part1, SfOutput_part2 are = The two parameters which are written to the csv file.

For exporting negative numbers, we used the single () command of Matlab, as follows:
brakeLightIndicator
=**single**(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
breaklightindicator)

MainLogBus. IncomingInterfaces. RacamObjData. Objects.Information.
breaklightindicator is an example where the parameters position are in the matlab structure.

3.3 Setting up the C++ testing environment

3.3.1 Using the Volvo program and setting up the Google Test

At the beginning of our work in setting up the C++ testing environment, we had the opportunity to help the Object Fusion team of Volvo and work together on a unit test case. This gave us a deeper understanding of the test functionalities, for our work.

3.3.2 Using Git and Gerrit

To manage code revisions and updates, in our project we used Git. We cloned the master branch of Volvo’s code platform for the Drive me project to access the last updated version of the master. This way, software developers and engineers at Volvo could still use the master file and make code updates parallel to our work.

3.3.3 Setting up the testing environment

For setting up our regression test environment, we used the Visual C++ Studio. However, after some weeks we realized that we could not see the Header files of object fusions, which was not optimal and became a big problem. Nevertheless, we discovered that we could use the object fusions test environment to get access to the necessary files for our test algorithm.

3.3.4 Setting up different ports

During the initial phase of our project, we learned how to use Google Test and how to construct fake ports with Google Mock. For allowing others understanding and accessing our code, we set up different ports in different structures to match what others do. We used these ports to implement the data from the .csv files and define the data in our C++ code so we could use them later in our testing environment for evaluation and comparison. Our C++ code implementing the data extracted from the .csv files has a similar structure as the Matlab file corresponding to the .csv file. Maps in Matlab are similar to C++ structs and our C++ code has been developed with many layers of advanced array structures to make code structuring and accessing easy and convenient.

3.3.5 An example of Google Test and Google Mock

Let us now illustrate the use of Google Test and Google Mock in our work.

Google Mock. Below is a Google Mock structure managing mock functions and using functions without being actually imported.

```
Class    Mock:  public class   // class name
{
Public:  // public are functions which you have access to outside the class
   MOCK_METHOD2 (funk1, funk2(par1,par2));
   MOCK_METHOD1 (run, funk3(par3));
};
#endif
```

Google Test. Below we show one of the Google Test functions where one can test whether the function was well invoked (has the right call). One can see that you call ‘run’ which is a function of the mock class and it is expected to return one. Usually this is used in a test case with many different function calls.

```
Mock mock();
EXPECT_CALL(mock,run(_)). willOnes(Return(1));
```

3.4 Designing and verifying the regression test algorithm

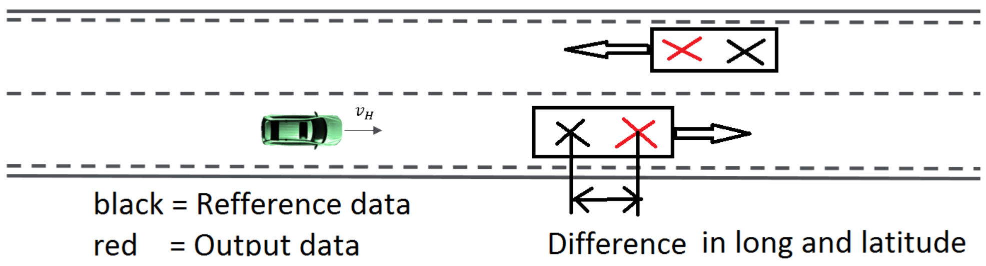


Figure 3:2 The difference of the position between the reference data and the output data.

Our testing algorithm is constructed so that the data from the sensors are from the scenario descried in Section 3.2, and the data is run through the Object Fusion functionality test. The Object Fusion functionality handles merging and adjusting object data from various sensors. Our algorithm performs a regression test against a reference received from the scenario formulated in a Matlab file. To ensure that regression does not to occur with changes in functionality, we checked and ensured that various object parameters from the output data do not deviate too much from the reference data.

Since the object list obtained as the output data from the Object Fusion functionality is not identical to the reference list, problems arise when these will be compared. The list of output has occasions when the objects change place or Id. With this problem in mind we created a ‘matching’-function to identify the items from output and reference lists with the most likely to belong together.

The ‘matching’ function described above are used in an algorithm that performs Pythagoras theorem to get the difference between the output and the reference position for each objects. See figure 3:2.

$$diff = sqar((ref\_long – out\_long)^2 + (ref\_lat – out\_lat)^2)$$

3.5 The Test modules functionality in C++

Here is a description of the test module and in appendix (see appendix B). The test module is constructed of three parts, namely initiation, functions calling and verifying.

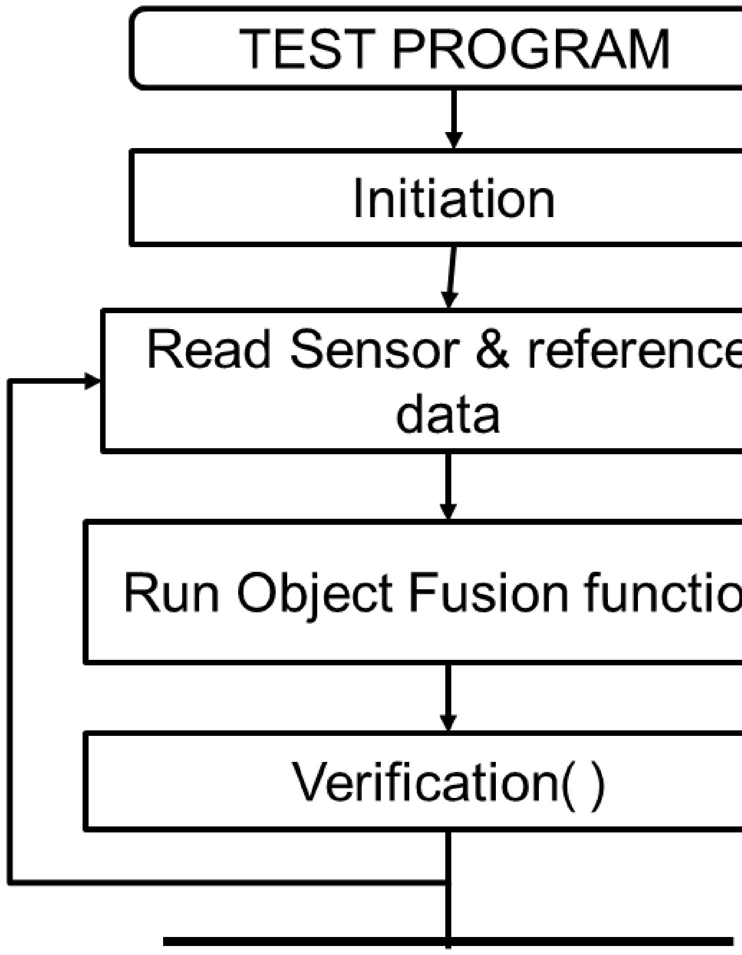


Figure 3:3 The Test Program functionality.

3.5.1 Initiation

Before the regression test can be executed, some conditions must be defined and controlled as follows:

- Constructing classes for reading the text files with sensor and reference data;
- Defining structures for the log data from the text files;
- Setting up the function from Object Fusion to test;
- Constructing the verification class;
- Do a test to control that the text files have the correct message count.

3.5.2 Call file reader and function for testing

For calling the Object fusion function module and the test verification method, we implemented the following steps:

- Read sensor data from text file;
- Read reference data that our test module will use to verify the function output with;
- Run the function module for testing;

- Run the test verification;
- These four stages are repeated for every time iteration.

3.5.3 Verifying the functions of output data

Figure 3:4 The Verification functionality.

In order to compare and evaluate the output and reference data for one iteration of time, we performed the following steps:

- Control of output vs reference data for amount of objects;
- Matching the nearest object, giving the correspondents of output and reference data;
- Analyse the parameters of object from the matching function:
  - position (difference in long/latitude with Pythagoras theorem)
  - speed
  - heading
  - class
  - ID
- Verification of the result from the analyses of the parameters and print to log file

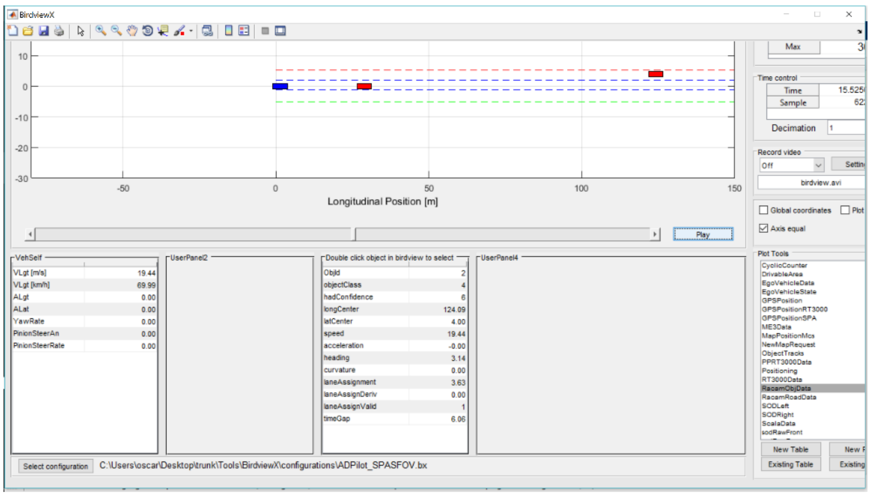
3.6 Visualising tests

To visualise our results and work progress during the project, we used two different approaches. When we want to show the concrete numbers of our testing scenarios, we used Excel. To visualize the scenarios, we used Birdview.

3.6.1 Visualisaton using Excel

In order to visualize the tests on the various differences that occur in the scenario built by the Google Test environment, we used Excel. This way, one can get a good overview of the parameters over time.

3.6.2 Visualisation using BirdView



VehSelf

VLgt [m/s]	19.44
VLgt [km/h]	69.99
ALgt	0.00
ALat	0.00
YawRate	0.00
PinionSteerAn	0.00
PinionSteerRate	0.00

Double click object in birdview to select

ObjId	2
objectClass	4
hadConfidence	6
longCenter	124.09
latCenter	4.00
speed	19.44
acceleration	-0.00
heading	3.14
curvature	0.00
laneAssignment	3.63
laneAssignDeriv	0.00
laneAssignValid	1
timeGap	6.06

Max	3

Time control

Time	15.525
Sample	62

Decimation 1

Record video: Off — birdview.avi

[ ] Global coordinates [ ] Plot
[✓] Axis equal

C:\Users\oscar\Desktop\trunk\Tools\BirdviewX\configurations\ADPilot_SPASFOV.bx

Figure 3:3 The BirdView program layout

We also used the BirdView program that Volvo developed in Matlab to get the opportunity to simulate data with different sensors from different scenarios and thus able to obtain the data that the car would have had if it were out of traffic with the same equipment.

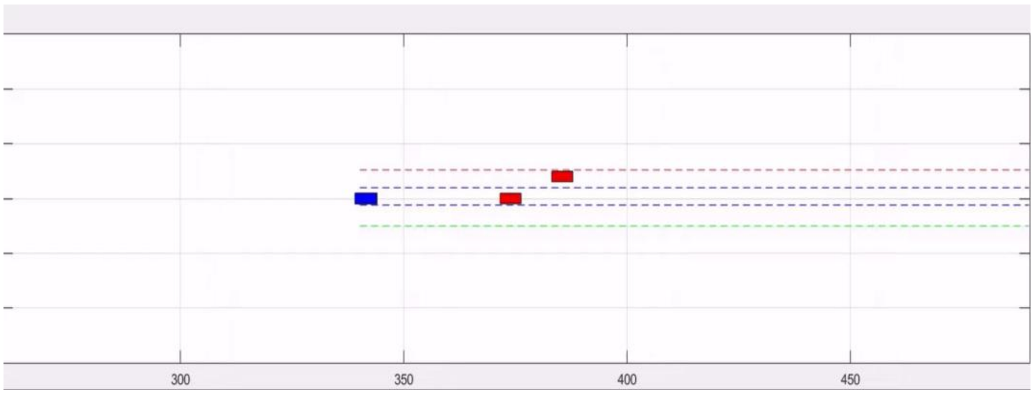


Figure 3:4 The BirdView program for the demo

Through BirdView, one can see the scenario and replay it, as well as see the different speeds of objects, the direction etc. BirdView is very user friendly which has allowed us to visualize our process during ‘sprint demos’ (see figure 3:4) and various meetings at Volvo.

4 Results

In our final solution we have used one matlab file for a scenario witch produce three csv files at 700 kb each. We have written about 500 lines of matlab code and 1500 lines of C++ code.

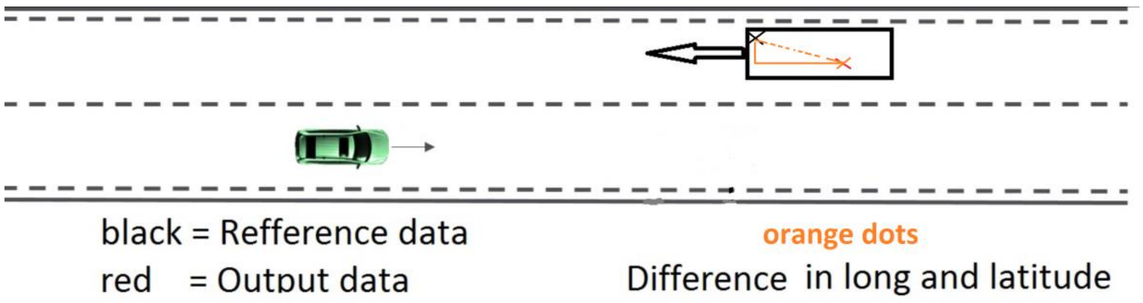


Figure 4:1 The use of the Pythagoras theorem
Here is a run of our distance algorithm through Pythagoras to compare the various diagonals between the reference data and the simulated output

Timestamp:		Difference before:		Difference after:
1800		0,801121		3,33432
1825		0,801121		3,33432
1850		0,801121		3,33432
1875		0,80112		3,33432
1900		0,80112		3,33431

Table 4:1 The results from two test times

Through these results it can be seen that the difference between the vehicle position and the reference point is a static error. This is consistent with what Volvo’s Object Fusion team previously mentioned, ensuring that our comparison algorithm and scenario is correct.

To the right in table 4.1 summarizes the results of our second run after object fusions have change the sensor structure and some other functionalities. The static error have increase a lot between these test runs and this is why tests like this regression test are needed. We know that none of these results are acceptable in the final software because the difference is supposed to be close to zero.

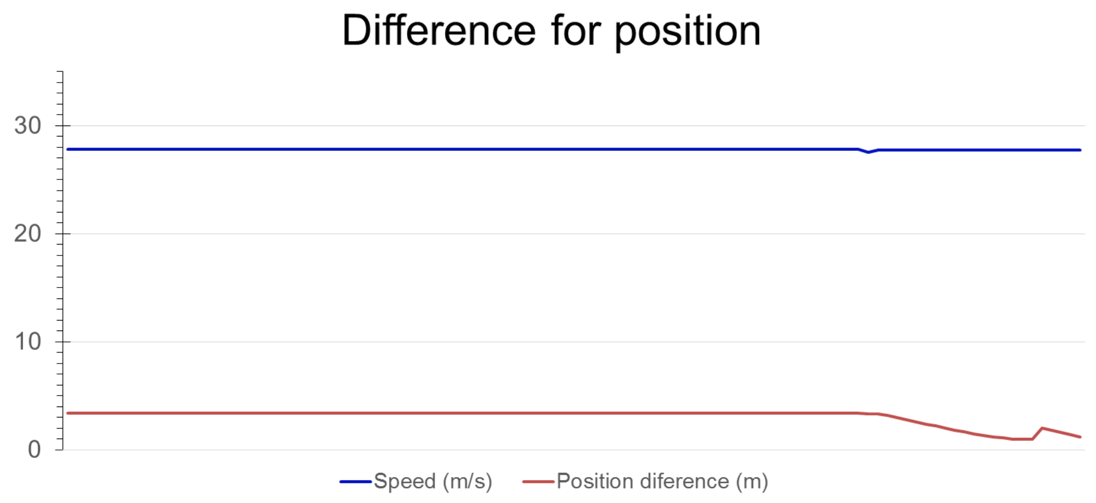


Figure 4:1 This is a graph of position difference over time for one of the vehicle for the start of the scenario.

In the final days of the thesis we plotted the poisoning difference. In Figure 4:1 you can see the result of one of the vehicles. We have not been able to define a reason for the drop of the error, this is something we recommend Volvo cars to dig deeper in.

5 Conclusion and experience

The current thesis studies functionalities of self-driving cars and brings the following contributions: (i) converting Matlab files implementing testing scenarios into C++ code to be used further in regression testing, and (ii) developing a regression test algorithm for comparing and evaluating different functionalities.

5.1 Converting the Matlab file

In the thesis we use a simulated scenario where one car overtake another one. This scenario is simulated using two sensors, the ‘Side Object Detector’ and the ‘Radar Camera’ sensors. This scenario is formulated in Matlab. To use the sensor data from these sensors, we first converted a Matlab file into C++ code. For doing so, we had to export the Matlab file with many structured arrays into text files, which posed challenges with exporting the arrays and structures of the Matlab files. Our solution was to export Matlab files to .csv files, avoiding any Matlab code dependencies. Our work also exported integer, floating and negatives numbers to the text files.

5.2 Algorithm / C++ code

The second contribution of our thesis is a regression test algorithm that automatically compares and evaluates the different parameters. To succeed to compare the output data from the Object Fusion data with the optimal simulated data on seven parameters, we used the difference between the reference data and output data for evaluating our result.

The difficult part was to compare the reference and output data and see if the compared data is within the range of a pass test or of a fail test. We found two different results when we run the same code. Once, the difference was around 0.8 meters and the next time 3.3 meters which a very large increase of the error which we do not yet know if it is a reasonable result and if it will pass the test or not.

5.3 Experience

5.3.1 Project experience

During our eight weeks we have not only got a great knowledge of how to test code , the software and the tests environment Volvo has chosen to select in this jungle of software.

We have also noticed how working in a large company is, where employees are heavily dependent on each other's work. We also experienced that solving problems becomes much easier with the help of various meetings and communications at the workplace.

When we started this thesis, our work plan was not completely clear. It was unclear how our project should proceed, despite that the overall end goal of regression testing was set. It was noticed quite quickly that we had to scale of the job, because it was difficult to keep up with all delays from different angle. During the project, we also depended on Volvo's Object Fusion team which should produce a scenario so that members of another team at Volvo could develop the log file for the sensor and output data. We needed that log file (as Matlab files) so that we could start designing and implementing our algorithm.

5.3.2 Code experience

We had to spend many hours at work to import the Matlab log files to C ++. This is because it was complicated to use Matlab's own library functions to get the right data in to our C ++ code.

During the preparation of the solution in the project we have met great programming challenges when the log structure and content had significantly changed.

With the development of our verification algorithm, we have worked with two alternative principles for matching objects between output from code and the reference from the simulated scenario. One matching was based on the coordinate data (longitude) by making a radius of the reference data who see objects within the radius and matches. The other matching was based on the nearest object in the reference list (longitude). We have chosen to match based on the nearest object in the list because this was easier this way to evaluate the functions when they changed.

5.4 Recommendations for further work

In the continued work of this algorithm / work, it is required to make a more overall product with all the sensors and many more parameters so that one knows that the different scenarios and their outputs are reasonable when compared with the reference case.

When more parameters and objects are used, the data exported to the csv files will yield larger files. Large files need however also be stored in the repository. Therefore, the question arises on how to keep file sizes small but still exporting and saving all relevant data.

We finally recommend to have many meetings with the developer teams so they do not change ports and structures more often than needed.

6 Reference

[1] Xie T, Kunal T, Kale S, Marinow M. Towards a Framework for Differential Unit Testing of Object-Oriented Programs [Internet]. 2012 [citerad 7 June 2016].

Hämtad från: http://people.engr.ncsu.edu/txie/publications/ast07-diffut.pdf

[2] Wikipedia. Regression testing [Internet]. 2016 [citerad 7 June 2016].

Hämtad från: https://en.wikipedia.org/wiki/Regression_testing

[3] Matlab [Internet]. 2012 [citerad 7 June 2015].

Hämtad från: http://se.mathworks.com/support/learn-with-matlab-tutorials.html

[4] Mathlab Support. Coder [Internet]. 2015 [citerad 7 June 2016].

Hämtad från: http://se.mathworks.com/matlabcentral/answers/223937-should-i-use-matlab-compiler-sdk-or-matlab-coder-to-integrate-my-matlab-applications-with-c-c

[5] Matio [Internet]. 2016 [citerad 7 June 2016].

Hämtad från: https://github.com/soumith/matio-ffi.torch

[6] Stack Overflow. Use Csv[Internet]. 2012 [citerad 7 June 2016].

Hämtad från: http://stackoverflow.com/questions/6479154/read-pixel-value-in-c-opencv'

[7] Mackinnon T. Mock [Internet]. 2009 [citerad 7 June 2016].

Hämtad från: http://www.mockobjects.com/2009/09/brief-history-of-mock-objects.html

[8] Preißel R, Stachmann B. Git: Distributed Version Control--Fundamentals and Workflows [Internet]. 2014 [citerad 7 June 2016].

Hämtad från: http://site.ebrary.com/lib/chalmers/reader.action?docID=10993744

[9] Rasmussen P. Gerrit [Internet]. 2011 [citerad 7 June 2016].

Hämtad från: https://review.openstack.org/Documentation/intro-quick.html

[10] Schwern M. Git For Ages 4 And Up [Internet]. 2013 [citerad 7 June 2016].

Hämtad från: https://www.youtube.com/watch?v=1ffBJ4sVUb4

[11] Whittaker, James (2012). How Google Tests Software. Boston, Massachusetts: Pearson Education. ISBN 0-321-80302-7.

[12] Google C++ Testing Framework.
https://github.com/google/googletest/blob/master/googletest/docs/Primer.md

[13] Different unit test framework

https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

A: Matlab kod för att generera csv filer

```
. function [] = GenerateDataCSV( logFullPath, saveFolder )
% generate_csv_files Function that extracts the sensor data from a
logFile and
% saves it to a .csv file containing all data.
% 2016-04-16

load(logFullPath,'MainLogBus');

% RaCam
timeStamp =
single(MainLogBus.IncomingInterfaces.RacamObjData.timeStamp);
timeStampIdn =
single(MainLogBus.IncomingInterfaces.RacamObjData.timeStampIdn);

% RaCam-objects-Estimate
longPosition =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.lon
gPosition);
latPosition =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.lat
Position);
stateThree =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.sta
teThree);
stateFour =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.sta
teFour);
stateFive =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.sta
teFive);
stateSix =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Estimate.sta
teSix);

% RaCam-objects-Information
type =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
type);
width =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
width);
height =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
height);
nearestSide =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
nearestSide);
turnIndicator =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
turnIndicator);
brakeLightIndicator =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
brakeLightIndicator);
hazardLightIndicator =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
hazardLightIndicator);
```

```
motionPatternCurrent =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
motionPatternCurrent);
motionPatternHistory =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Information.
motionPatternHistory);

% RaCam-objects-Properties
id =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.i
d);
trackStatus =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.t
rackStatus);
motionModel =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.m
otionModel);
tjaConfidence =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.t
jaConfidence);
trackOrigin =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.t
rackOrigin);
longPositionStdDev =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.l
ongPositionStdDev);
latPositionStdDev =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.l
atPositionStdDev);
headingStdDev =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.h
eadingStdDev);
speedStdDev =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.s
peedStdDev);
accelerationStdDev =
single(MainLogBus.IncomingInterfaces.RacamObjData.Objects.Properties.a
ccelerationStdDev);

% Samanställning Racam
Racam_time = [timeStamp,timeStampIdn];
Racam_estimate =
[longPosition,latPosition,stateThree,stateFour,stateFive,stateSix];
Racam_information =
[type,width,height,nearestSide,turnIndicator,brakeLightIndicator,hazar
dLightIndicator,motionPatternCurrent,motionPatternHistory];
Racam_Properties =
[id,trackStatus,motionModel,tjaConfidence,trackOrigin,longPositionStdD
ev,latPositionStdDev,headingStdDev,speedStdDev,accelerationStdDev];
```

```
% SOD-Left
SODLeft_timeStampSim =
single(MainLogBus.IncomingInterfaces.SODLeft.timeStampSim);
SODLeft_versionInfo =
single(MainLogBus.IncomingInterfaces.SODLeft.versionInfo);
SODLeft_sourceTxCnt =
single(MainLogBus.IncomingInterfaces.SODLeft.sourceTxCnt);
SODLeft_sourceTxTime =
single(MainLogBus.IncomingInterfaces.SODLeft.sourceTxTime);
SODLeft_sourceInfo =
single(MainLogBus.IncomingInterfaces.SODLeft.sourceInfo);
SODLeft_reservedSrc1 =
single(MainLogBus.IncomingInterfaces.SODLeft.reservedSrc1);
SODLeft_reservedSrc2 =
single(MainLogBus.IncomingInterfaces.SODLeft.reservedSrc2);
SODLeft_reservedSrc3 =
single(MainLogBus.IncomingInterfaces.SODLeft.reservedSrc3);
SODLeft_streamRefIndex =
single(MainLogBus.IncomingInterfaces.SODLeft.streamRefIndex);
SODLeft_streamDataLen =
single(MainLogBus.IncomingInterfaces.SODLeft.streamDataLen);
SODLeft_streamTxCnt =
single(MainLogBus.IncomingInterfaces.SODLeft.streamTxCnt);
SODLeft_streamNumber =
single(MainLogBus.IncomingInterfaces.SODLeft.streamNumber);
SODLeft_streamVersion =
single(MainLogBus.IncomingInterfaces.SODLeft.streamVersion);
SODLeft_streamChunks =
single(MainLogBus.IncomingInterfaces.SODLeft.streamChunks);
SODLeft_streamChunkIdx =
single(MainLogBus.IncomingInterfaces.SODLeft.streamChunkIdx);
SODLeft_reservedStr3 =
single(MainLogBus.IncomingInterfaces.SODLeft.reservedStr3);
SODLeft_streamID =
single(MainLogBus.IncomingInterfaces.SODLeft.streamID);
SODLeft_maTimeStamp =
single(MainLogBus.IncomingInterfaces.SODLeft.maTimeStamp);
SODLeft_slTimeStamp =
single(MainLogBus.IncomingInterfaces.SODLeft.slTimeStamp);
SODLeft_LongDistRearAxle =
single(MainLogBus.IncomingInterfaces.SODLeft.LongDistRearAxle);
SODLeft_maScIndex =
single(MainLogBus.IncomingInterfaces.SODLeft.maScIndex);
SODLeft_slScIndex =
single(MainLogBus.IncomingInterfaces.SODLeft.slScIndex);
SODLeft_minorVersion =
single(MainLogBus.IncomingInterfaces.SODLeft.minorVersion);
SODLeft_majorVersion =
single(MainLogBus.IncomingInterfaces.SODLeft.majorVersion);
SODLeft_packetSize =
single(MainLogBus.IncomingInterfaces.SODLeft.packetSize);
SODLeft_noTrackerObj =
single(MainLogBus.IncomingInterfaces.SODLeft.noTrackerObj);
SODLeft_id = single(MainLogBus.IncomingInterfaces.SODLeft.id);
SODLeftlonPo_ = single(MainLogBus.IncomingInterfaces.SODLeft.lonPo);
SODLeft_lonVel = single(MainLogBus.IncomingInterfaces.SODLeft.lonVel);
SODLeft_lonAcc = single(MainLogBus.IncomingInterfaces.SODLeft.lonAcc);
SODLeft_latPos = single(MainLogBus.IncomingInterfaces.SODLeft.latPos);
SODLeft_latVel = single(MainLogBus.IncomingInterfaces.SODLeft.latVel);
```

```
SODLeft_latAcc = single(MainLogBus.IncomingInterfaces.SODLeft.latAcc);
SODLeft_status = single(MainLogBus.IncomingInterfaces.SODLeft.status);
SODLeft_heading =
single(MainLogBus.IncomingInterfaces.SODLeft.heading);
SODLeft_length = single(MainLogBus.IncomingInterfaces.SODLeft.length);
SODLeft_width = single(MainLogBus.IncomingInterfaces.SODLeft.width);
SODLeft_refPosition =
single(MainLogBus.IncomingInterfaces.SODLeft.refPosition);
SODLeft_objectType =
single(MainLogBus.IncomingInterfaces.SODLeft.objectType);
SODLeft_existenceProb =
single(MainLogBus.IncomingInterfaces.SODLeft.existenceProb);


% Samanställning SOD-Left
SODLeft_part1 =
[SODLeft_timeStampSim,SODLeft_versionInfo,SODLeft_sourceTxCnt,SODLeft_
sourceTxTime,SODLeft_sourceInfo,SODLeft_reservedSrc1,SODLeft_reservedS
rc2,SODLeft_reservedSrc3,SODLeft_streamRefIndex,SODLeft_streamDataLen]
;
SODLeft_part2 =
[SODLeft_streamTxCnt,SODLeft_streamNumber,SODLeft_streamVersion,SODLef
t_streamChunks,SODLeft_streamChunkIdx,SODLeft_reservedStr3,SODLeft_str
eamID,SODLeft_maTimeStamp,SODLeft_slTimeStamp,SODLeft_LongDistRearAxle
];
SODLeft_part3 =
[SODLeft_maScIndex,SODLeft_slScIndex,SODLeft_minorVersion,SODLeft_majo
rVersion,SODLeft_packetSize,SODLeft_noTrackerObj,SODLeft_id,SODLeftlon
Po_,SODLeft_lonVel,SODLeft_lonAcc];
SODLeft_part4 =
[SODLeft_latPos,SODLeft_latVel,SODLeft_latAcc,SODLeft_status,SODLeft_h
eading,SODLeft_length,SODLeft_width,SODLeft_refPosition,SODLeft_object
Type,SODLeft_existenceProb];


% SOD-Right
SODRight_timeStampSim =
single(MainLogBus.IncomingInterfaces.SODRight.timeStampSim);
SODRight_versionInfo =
single(MainLogBus.IncomingInterfaces.SODRight.versionInfo);
SODRight_sourceTxCnt =
single(MainLogBus.IncomingInterfaces.SODRight.sourceTxCnt);
SODRight_sourceTxTime =
single(MainLogBus.IncomingInterfaces.SODRight.sourceTxTime);
SODRight_sourceInfo =
single(MainLogBus.IncomingInterfaces.SODRight.sourceInfo);
SODRight_reservedSrc1 =
single(MainLogBus.IncomingInterfaces.SODRight.reservedSrc1);
SODRight_reservedSrc2 =
single(MainLogBus.IncomingInterfaces.SODRight.reservedSrc2);
SODRight_reservedSrc3 =
single(MainLogBus.IncomingInterfaces.SODRight.reservedSrc3);
SODRight_streamRefIndex =
single(MainLogBus.IncomingInterfaces.SODRight.streamRefIndex);
SODRight_streamDataLen =
single(MainLogBus.IncomingInterfaces.SODRight.streamDataLen);
SODRight_streamTxCnt =
single(MainLogBus.IncomingInterfaces.SODRight.streamTxCnt);
```

```
SODRight_streamNumber =
single(MainLogBus.IncomingInterfaces.SODRight.streamNumber);
SODRight_streamVersion =
single(MainLogBus.IncomingInterfaces.SODRight.streamVersion);
SODRight_streamChunks =
single(MainLogBus.IncomingInterfaces.SODRight.streamChunks);
SODRight_streamChunkIdx =
single(MainLogBus.IncomingInterfaces.SODRight.streamChunkIdx);
SODRight_reservedStr3 =
single(MainLogBus.IncomingInterfaces.SODRight.reservedStr3);
SODRight_streamID =
single(MainLogBus.IncomingInterfaces.SODRight.streamID);
SODRight_maTimeStamp =
single(MainLogBus.IncomingInterfaces.SODRight.maTimeStamp);
SODRight_slTimeStamp =
single(MainLogBus.IncomingInterfaces.SODRight.slTimeStamp);
SODRight_LongDistRearAxle =
single(MainLogBus.IncomingInterfaces.SODRight.LongDistRearAxle);
SODRight_maScIndex =
single(MainLogBus.IncomingInterfaces.SODRight.maScIndex);
SODRight_slScIndex =
single(MainLogBus.IncomingInterfaces.SODRight.slScIndex);
SODRight_minorVersion =
single(MainLogBus.IncomingInterfaces.SODRight.minorVersion);
SODRight_majorVersion =
single(MainLogBus.IncomingInterfaces.SODRight.majorVersion);
SODRight_packetSize =
single(MainLogBus.IncomingInterfaces.SODRight.packetSize);
SODRight_noTrackerObj =
single(MainLogBus.IncomingInterfaces.SODRight.noTrackerObj);
SODRight_id = single(MainLogBus.IncomingInterfaces.SODRight.id);
SODRight_lonPo = single(MainLogBus.IncomingInterfaces.SODRight.lonPo);
SODRight_lonVel =
single(MainLogBus.IncomingInterfaces.SODRight.lonVel);
SODRight_lonAcc =
single(MainLogBus.IncomingInterfaces.SODRight.lonAcc);
SODRight_latPos =
single(MainLogBus.IncomingInterfaces.SODRight.latPos);
SODRight_latVel =
single(MainLogBus.IncomingInterfaces.SODRight.latVel);
SODRight_latAcc =
single(MainLogBus.IncomingInterfaces.SODRight.latAcc);
SODRight_status =
single(MainLogBus.IncomingInterfaces.SODRight.status);
SODRight_heading =
single(MainLogBus.IncomingInterfaces.SODRight.heading);
SODRight_length =
single(MainLogBus.IncomingInterfaces.SODRight.length);
SODRight_width = single(MainLogBus.IncomingInterfaces.SODRight.width);
SODRight_refPosition =
single(MainLogBus.IncomingInterfaces.SODRight.refPosition);
SODRight_objectType =
single(MainLogBus.IncomingInterfaces.SODRight.objectType);
SODRight_existenceProb =
single(MainLogBus.IncomingInterfaces.SODRight.existenceProb);
```

```
% Samanställning SOD-Right
SODRight_part1 =
[SODRight_timeStampSim,SODRight_versionInfo,SODRight_sourceTxCnt,SODRi
ght_sourceTxTime,SODRight_sourceInfo,SODRight_reservedSrc1,SODRight_re
servedSrc2,SODRight_reservedSrc3,SODRight_streamRefIndex,SODRight_stre
amDataLen];
SODRight_part2 =
[SODRight_streamTxCnt,SODRight_streamNumber,SODRight_streamVersion,SOD
Right_streamChunks,SODRight_streamChunkIdx,SODRight_reservedStr3,SODRi
ght_streamID,SODRight_maTimeStamp,SODRight_slTimeStamp,SODRight_LongDi
stRearAxle];
SODRight_part3 =
[SODRight_maScIndex,SODRight_slScIndex,SODRight_minorVersion,SODRight_
majorVersion,SODRight_packetSize,SODRight_noTrackerObj,SODRight_id,SOD
Right_lonPo,SODRight_lonVel,SODRight_lonAcc];
SODRight_part4 =
[SODRight_latPos,SODRight_latVel,SODRight_latAcc,SODRight_status,SODRi
ght_heading,SODRight_length,SODRight_width,SODRight_refPosition,SODRig
ht_objectType,SODRight_existenceProb];


% SfOutput - position
SfOutput_position_state = MainLogBus.SfOutput.Positioning.states;
SfOutput_xMcs = MainLogBus.SfOutput.Positioning.xMcs;
SfOutput_pathIndex = MainLogBus.SfOutput.Positioning.pathIndex;
SfOutput_yMcs = MainLogBus.SfOutput.Positioning.yMcs;
SfOutput_zMcs = MainLogBus.SfOutput.Positioning.zMcs;
SfOutput_phiMcs = MainLogBus.SfOutput.Positioning.phiMcs;
SfOutput_polyOffset= MainLogBus.SfOutput.Positioning.polyOffset;
SfOutput_polyId = MainLogBus.SfOutput.Positioning.polyId;
SfOutput_isValid = MainLogBus.SfOutput.Positioning.isValid;
SfOutput_isInitiated = MainLogBus.SfOutput.Positioning.isInitiated;
SfOutput_laneChange = MainLogBus.SfOutput.Positioning.laneChange;
SfOutput_resetFilter = MainLogBus.SfOutput.Positioning.resetFilter;
SfOutput_covariance = MainLogBus.SfOutput.Positioning.covariance;

% SfOutput - objectTracks
SfOutput_timerStamp =
single(MainLogBus.SfOutput.ObjectTracks.timeStamp);

% SfOutput - objectTracks - states
SfOutput_longPos =
single(MainLogBus.SfOutput.ObjectTracks.States.longPos);
SfOutput_latPos =
single(MainLogBus.SfOutput.ObjectTracks.States.latPos);
SfOutput_States_heading =
single(MainLogBus.SfOutput.ObjectTracks.States.heading);
SfOutput_speed =
single(MainLogBus.SfOutput.ObjectTracks.States.speed);
SfOutput_curvature =
single(MainLogBus.SfOutput.ObjectTracks.States.curvature);
SfOutput_acceleration =
single(MainLogBus.SfOutput.ObjectTracks.States.acceleration);
```

```
%SfOutput - objectTracks - Box
SfOutput_longcenter =
single(MainLogBus.SfOutput.ObjectTracks.Box.longCenter);
SfOutput_latcenter =
single(MainLogBus.SfOutput.ObjectTracks.Box.latCenter);
SfOutput_Box_heading =
single(MainLogBus.SfOutput.ObjectTracks.Box.heading);
SfOutput_width = single(MainLogBus.SfOutput.ObjectTracks.Box.width);
SfOutput_length = single(MainLogBus.SfOutput.ObjectTracks.Box.length);
SfOutput_height = single(MainLogBus.SfOutput.ObjectTracks.Box.height);

%SfOutput - objectTracks - Attrib
SfOutput_id = single(MainLogBus.SfOutput.ObjectTracks.Attrib.id);
SfOutput_status =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.status);
SfOutput_objectClass =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.objectClass);
SfOutput_turnIndicator =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.turnIndicator);
SfOutput_hadConfidence =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.hadConfidence);
SfOutput_brakeLightIndicator =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.brakeLightIndicator);
SfOutput_hazardLightIndicator =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.hazardLightIndicator);
SfOutput_motionPattern =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.motionPattern);
SfOutput_motionPatternHistory =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.motionPatternHistory);
SfOutput_refPointLocation =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.refPointLocation);
SfOutput_fusedSensors =
single(MainLogBus.SfOutput.ObjectTracks.Attrib.fusedSensors);

%SfOutput - objectTracks - LaneAssignment
SfOutput_OffsetValid =
single(MainLogBus.SfOutput.ObjectTracks.LaneAssignment.lateralCenterOf
fsetValid);
SfOutput_CenterOffset =
single(MainLogBus.SfOutput.ObjectTracks.LaneAssignment.lateralCenterOf
fset);
SfOutput_OffsetDerivative =
single(MainLogBus.SfOutput.ObjectTracks.LaneAssignment.lateralCenterOf
fsetDerivative);

%SfOutput - DrivableArea - Current
SfOutput_Current_type = MainLogBus.SfOutput.DrivableArea.Current.type;
SfOutput_Current_leftLongitudinalBoundary =
MainLogBus.SfOutput.DrivableArea.Current.leftLongitudinalBoundary;
SfOutput_Current_leftLateralBoundary =
MainLogBus.SfOutput.DrivableArea.Current.leftLateralBoundary;
SfOutput_Current_leftBoundaryValid =
MainLogBus.SfOutput.DrivableArea.Current.leftBoundaryValid;
SfOutput_Current_rightLongitudinalBoundary =
MainLogBus.SfOutput.DrivableArea.Current.rightLongitudinalBoundary;
SfOutput_Current_rightLateralBoundary =
MainLogBus.SfOutput.DrivableArea.Current.rightLateralBoundary;
```

```
SfOutput_Current_rightBoundaryValid =
MainLogBus.SfOutput.DrivableArea.Current.rightBoundaryValid;
SfOutput_Current_roadCenterCurvature =
MainLogBus.SfOutput.DrivableArea.Current.roadCenterCurvature;
SfOutput_Current_roadCenterHeading =
MainLogBus.SfOutput.DrivableArea.Current.roadCenterHeading;
SfOutput_Current_roadCenterValid =
MainLogBus.SfOutput.DrivableArea.Current.roadCenterValid;

SfOutput_laneChange = MainLogBus.SfOutput.DrivableArea.laneChange;

% SfOutput - DrivableArea
SfOutput_DrivableAreaActive =
MainLogBus.SfOutput.DrivableArea.isDrivableAreaActive;

% SfOutput - EgoVehicleState
SfOutput_SpeedLong =
MainLogBus.SfOutput.EgoVehicleState.timeStampVehicleSpeedLong;

%SfOutput - EgoVehicleState - VehicleSpeedLong
SfOutput_longSpeed =
MainLogBus.SfOutput.EgoVehicleState.VehicleSpeedLong.longSpeed;
SfOutput_SpeedQuality =
MainLogBus.SfOutput.EgoVehicleState.VehicleSpeedLong.longSpeedQuality;

%SfOutput - EgoVehicleState
SfOutput_AccelerationLong =
MainLogBus.SfOutput.EgoVehicleState.timeStampVehicleAccelerationLong;

%SfOutput - EgoVehicleState - VehicleAcc
SfOutput_longAcceleration =
MainLogBus.SfOutput.EgoVehicleState.VehicleAccelerationLong.longAccele
ration;
SfOutput_AccelerationQuality =
MainLogBus.SfOutput.EgoVehicleState.VehicleAccelerationLong.longAccele
rationQuality;

%SfOutput - EgoVehicleState
SfOutput_VehicleAccelerationLat =
MainLogBus.SfOutput.EgoVehicleState.timeStampVehicleAccelerationLat;

%SfOutput - EgoVehicleState - vehicleYawRate
SfOutput_RollRate =
MainLogBus.SfOutput.EgoVehicleState.VehicleYawRate.RollRate;
SfOutput_RollRateQuality =
MainLogBus.SfOutput.EgoVehicleState.VehicleYawRate.RollRateQuality;
SfOutput_YawRate =
MainLogBus.SfOutput.EgoVehicleState.VehicleYawRate.YawRate;
SfOutput_YawRateQuality =
MainLogBus.SfOutput.EgoVehicleState.VehicleYawRate.YawRateQuality;

SfOutput_PinionSteerAngle_PinionSteerAngle =
MainLogBus.SfOutput.EgoVehicleState.PinionSteerAngle.PinionSteerAngle;
SfOutput_PinionSteerAngle_PinionSteerAngleQuality =
MainLogBus.SfOutput.EgoVehicleState.PinionSteerAngle.PinionSteerAngleQ
uality;
```

```
%SfOutput - EgoVehicleState
SfOutput_timeStampPinionSteerRate =
MainLogBus.SfOutput.EgoVehicleState.timeStampPinionSteerRate;

%SfOutput - EgoVehicleState - PinionSteerRate
SfOutput_PinionSteerRate.PinionSteerRate =
MainLogBus.SfOutput.EgoVehicleState.PinionSteerRate.PinionSteerRate;
SfOutput_PinionSteerRate.PinionSteerRateQuality =
MainLogBus.SfOutput.EgoVehicleState.PinionSteerRate.PinionSteerRateQua
lity;

%SfOutput - EgoVehicleState
SfOutput_deltaLongPosition =
MainLogBus.SfOutput.EgoVehicleState.deltaLongPosition;
SfOutput_deltaLatPosition =
MainLogBus.SfOutput.EgoVehicleState.deltaLatPosition;
SfOutput_deltaHeading =
MainLogBus.SfOutput.EgoVehicleState.deltaHeading;

% SfOutput
SfOutput_CyclicCounter = MainLogBus.SfOutput.CyclicCounter;
SfOutput_NewMapRequest = MainLogBus.SfOutput.NewMapRequest;

% sammanställning
%SfOutput_part1 =
[SfOutput_position_state,SfOutput_xMcs,SfOutput_pathIndex,SfOutput_yMc
s,SfOutput_zMcs,SfOutput_phiMcs,SfOutput_polyOffset,SfOutput_polyId,Sf
Output_isValid,SfOutput_isInitiated,SfOutput_laneChange,SfOutput_reset
Filter,SfOutput_covariance];

SfOutput_part2 =
[SfOutput_timerStamp,SfOutput_longPos,SfOutput_latPos,SfOutput_States_
heading,SfOutput_speed,SfOutput_curvature,SfOutput_acceleration,SfOutp
ut_longcenter,SfOutput_latcenter,SfOutput_Box_heading,SfOutput_width,S
fOutput_length,SfOutput_height];
SfOutput_part3 =
[SfOutput_id,SfOutput_status,SfOutput_objectClass,SfOutput_turnIndicat
or,SfOutput_hadConfidence,SfOutput_brakeLightIndicator,SfOutput_hazard
LightIndicator,SfOutput_motionPattern,SfOutput_motionPatternHistory,Sf
Output_refPointLocation,SfOutput_fusedSensors,SfOutput_OffsetValid,SfO
utput_CenterOffset,SfOutput_OffsetDerivative];

%SfOutput_part4 =
[SfOutput_Current_type,SfOutput_Current_leftLongitudinalBoundary,SfOut
put_Current_leftLateralBoundary,SfOutput_Current_leftBoundaryValid,SfO
utput_Current_rightLongitudinalBoundary,SfOutput_Current_rightLateralB
oundary,SfOutput_Current_rightBoundaryValid,SfOutput_Current_roadCente
rCurvature,SfOutput_Current_roadCenterHeading,SfOutput_Current_roadCen
terValid,SfOutput_AdjacentLeft_type,SfOutput_AdjacentLeft_leftLongitud
inalBoundary,SfOutput_AdjacentLeft_leftLateralBoundary,SfOutput_Adjace
ntLeft_leftBoundaryValid,SfOutput_AdjacentLeft_rightLongitudinalBounda
ry,SfOutput_AdjacentLeft_rightLateralBoundary,SfOutput_AdjacentLeft_ri
ghtBoundaryValid,SfOutput_AdjacentLeft_roadCenterCurvature,SfOutput_Ad
jacentLeft_roadCenterHeading,SfOutput_AdjacentLeft_roadCenterValid];
%SfOutput_part5 =
[SfOutput_AdjacentRight_type,SfOutput_AdjacentRight_leftLongitudinalBo
undary,SfOutput_AdjacentRight_leftLateralBoundary,SfOutput_AdjacentRig
ht_leftBoundaryValid,SfOutput_AdjacentRight_rightLongitudinalBoundary,
SfOutput_AdjacentRight_rightLateralBoundary,SfOutput_AdjacentRight_rig
htBoundaryValid,SfOutput_AdjacentRight_roadCenterCurvature,SfOutput_Ad
jacentRight_roadCenterHeading,SfOutput_AdjacentRight_roadCenterValid,S
fOutput_laneChange,SfOutput_DrivableAreaActive,SfOutput_SpeedLong];
%SfOutput_part6 =
[SfOutput_longSpeed,SfOutput_SpeedQuality,SfOutput_AccelerationLong,Sf
Output_longAcceleration,SfOutput_AccelerationQuality,SfOutput_VehicleA
ccelerationLat,SfOutput_RollRate,SfOutput_RollRateQuality,SfOutput_Yaw
Rate,SfOutput_YawRateQuality,SfOutput_timeStampPinionSteerAngle,SfOutp
ut_PinionSteerAngle_PinionSteerAngle,SfOutput_PinionSteerAngle_PinionS
teerAngleQuality,SfOutput_timeStampPinionSteerRate,SfOutput_PinionStee
rRate.PinionSteerRateQuality,SfOutput_deltaLongPosition,SfOutput_delta
LatPosition,SfOutput_deltaHeading,SfOutput_CyclicCounter,SfOutput_NewM
apRequest];



[logFolder, logName, ~] = fileparts(logFullPath);

if(nargin > 1)
    % savePath = fullfile(saveFolder, ['ObjData_', logName, '.csv']);
    Racam_savePath = fullfile(saveFolder, 'ObjData_RaCam.csv');
    SODLeft_savePath = fullfile(saveFolder, 'ObjData_SODLeft.csv');
    SODRight_savePath = fullfile(saveFolder, 'ObjData_SODRight.csv');
    SfOutput_savePath = fullfile(saveFolder, 'SfOutput.csv');
else
    % savePath = fullfile(logFolder, ['ObjData_', logName, '.csv']);
    Racam_savePath = fullfile(logFolder, 'ObjData_RaCam.csv');
    SODLeft_savePath = fullfile(logFolder, 'ObjData_SODLeft.csv');
    SODRight_savePath = fullfile(logFolder, 'ObjData_SODRight.csv');
    SfOutput_savePath = fullfile(logFolder, 'ObjData_SfOutput.csv');
end
    % wright obejects to file
    dlmwrite(Racam_savePath,
[Racam_time,Racam_estimate,Racam_information,Racam_Properties],'precis
ion','%.4f');
    dlmwrite(SODLeft_savePath,
[SODLeft_part1,SODLeft_part2,SODLeft_part3,SODLeft_part4],'precision',
'%.4f');

%[SODLeft_latAcc,SODLeft_timeStampSim,SODLeft_lonVel,SODLeft_versionIn
fo,SODLeft_sourceTxCnt,SODLeft_sourceTxTime,SODLeft_sourceInfo,SODLeft
_reservedSrc1,SODLeft_reservedSrc2,SODLeft_reservedSrc3,SODLeft_stream
RefIndex,SODLeft_streamDataLen,SODLeft_streamTxCnt,SODLeft_streamNumbe
r,SODLeft_streamVersion,SODLeft_streamChunks],'precision','%.4f');

    %dlmwrite(SODRight_savePath,
[SODRight_part1,SODRight_part2,SODRight_part3,SODRight_part4],'precisi
on','%.4f');
    dlmwrite(SfOutput_savePath,
[SfOutput_part2,SfOutput_part3],'precision','%.4f');

    %dlmwrite(SfOutput_savePath,
[SfOutput_part1,SfOutput_part2,SfOutput_part3,SfOutput_part4,SfOutput_
part5,SfOutput_part6]);
end
```