POLITECNICO DI TORINO

DIMEAS

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

DESIGNING OF A MATLAB APPLICATION TO SPEED UP THE GOLDEN PASS TEST SELECTION IN DURABILITY ANALYSIS OF VEHICLES

SUPERVISORS:

Prof. PAOLINO DAVIDE SALVATORE

Prof. CIAMPAGLIA ALBERTO

TUTOR:

Eng. BRUNO GIUSEPPE

CANDIDATE:

MELILLO DOMENICO

Academic year 2023-2024

ABSTRACT

The aim of this thesis is to drastically change the method adopted for the selection of the Golden Pass test among several performed during the durability analysis of a vehicle in the STELLANTIS group. The Golden Pass is characterized as the test closer, fatigue wise, to the solicitations a vehicle would undergo in a period of its life; the test is then replicated in a more efficient way on the proving benches.

In the durability analysis department of STELLANTIS, the fatigue prediction method which is commonly adopted is the level crossing one, which compromises the accuracy in prediction with the ease of reading the results.

In the company, the Golden Pass test is manually selected after a punctual analysis of each level crossing graph concerning every accelerometer present on a test car, depicting the most accurate one. This process besides being extremely time consuming is prone to major errors that could arise due to poor graphical resolution of a graph or to the human error and distraction.

The employment of a MATLAB script would both greatly accelerate the process and avoid minor errors that could arise otherwise; at the same time graphs will be produced by the script as well so that in case of doubts on the obtained results the technician can promptly analyze a single channel's graph and check if any mistake is present without going through the entire channels' database.

The application is designed to produce a summary table depicting the Golden Pass test on an objective scale using marks in order to produce a more intuitive to read and easier to share result across the company.

1. INTRODUCTION

Fatigue problems are characterized by the fact that repeated stresses can cause a fracture after some time and fatigue analysis uses this phenomenon in order to predict the life of a component.

There are many methods to predict life lengths. However, the relationship between the life lengths of components exposed to regular oscillating stresses and the life lengths of components exposed to more irregular loads, like it is in a complex structure as a vehicle, is not clear.

The objective in the automotive industry is to test as few prototypes as possible; preferably only one of any finalized models. Although, at present, we have to create and verify several more design and simulation software before we can get to this ideal, the objective now appears attainable. The usual economic forces of reducing numbers of prototypes, minimization of testing and design iterations, optimizing gages and weight, and shortened design cycle time are driving us toward this objective.

Therefore, the testing on those few prototypes has to maximize efficiency and verify every assumption done in the design phase and every software simulation. This kind of testing is called durability analysis.

1.1. Durability analysis

Automotive durability testing supports component and original component manufacturers' evaluation of vehicles, subsystems and components to determine expected life.

Tests use real-world motions and forces to assess and validate design and improve quality. In addition to complete vehicles, automotive durability testing evaluates components including exhaust. chassis, frame, trailer hitches, seating, cooling systems, interiors, and suspension.

Conducting mechanical, electrical and electro-mechanical automotive durability testing provides insights to determine expected product life and identify areas for improving performance.

This thesis focuses its attention on accelerated testing on endurance and reliability tracks. Measurements of loads and fatigue on normal and heavy-duty routes and in general all the mechanical tests which must be performed on a vehicle according to company's standards.

The test standards offer the option to verify the structural strength of an entire vehicle, the transmission and all aspects related to corrosion. Vehicles can be fatigued for a few months on the special routes, to represent their entire lifespan.

There are several types of durability test:

1.1.1. General durability

Vehicle durability testing involves driving the vehicle and subjecting it to inputs similar to those normally encountered during its expected lifetime. It is easy to find photos and videos of camouflaged vehicles on highways and other roads. They can usually take a year to complete. In most cases, these vehicles travel more than a thousand kilometers during the test.[1]

1.1.2. Accelerated durability test

Accelerated durability life tests are designed to quantify problems that could occur in 10-15 years of a vehicle by testing at a higher stress level to accelerate the occurrence of failure in as little as three months. These tests are conducted on specially prepared tracks that are designed to contain various surfaces including bumps, cobblestones, resonance, undulating roads, etc. A four-hour drive on these tracks would be equivalent to driving 1000 km on normal roads.[1]

1.1.3. MAST

Multi-The Axis Shaking Table, or MAST, is a dynamic system designed to perform R&D tests at the system level. (System level means a group of components that must work together, e.g. dashboard, seats). The MAST system moves up, down and sideways at a very high frequency. MAST can help avoid testing the entire vehicle for small parts. MAST can also analyze the operation of seats, doors and switches. There is a device designed to detect problems that could arise over the years in various components. It was possible to simulate 10-15 years of use within days.[1]

1.2. Level crossing

One of the common models to carry out a fatigue analysis, is the level crossing approach. In this case the stress history is specified by the number of level crossings and the total damage is calculated by mean of the following integral.

$$D = \int_{-\infty}^{+\infty} N(s) \cdot g(s)\, ds$$

In the equation, D is a measure of damage, N(s) is the number of up crossings at stress level s, and g is an exhaustion density function, derived from constant amplitude results.

A relevant downside of the level crossing approach is that it assumes that the damage is independent from the order in which the stress intervals of arbitrary length appear and in the corresponding model intervals are mixed.[2]

Higher degree of mixing in the level crossing model would accordingly give less accurate lifetime predictions.

The assumption of order independence could also be interpreted as a lack of memory in the fatigue damage process. The level crossing model contains in this sense no memory at all.

In most laboratory experiments the Rain flow counting technique has been found superior to all other methods, included level crossing. However, in many investigations it has been shown that even the Rain Flow counting method gives non-conservative predictions and in some of these cases actually the level crossing model is superior; this seem to be the case when the stress process is considerably irregular.[3]

2. DURABILITY ANALYSIS TESTS DESCRIPTION

2.1. Balocco proving ground

2.1.1. Different schedules

The STELLANTIS group has several proving grounds each with its relative standards, this thesis has been focusing on the durability tests performed in Balocco proving ground situated in the province of Vercelli, Italy.

Following the standards of the company, the facilities in Balocco proving ground are able to accommodate durability tests for several kinds of vehicles, each with its own schedule ranging from light commercial vehicles to extreme sport cars.

Among the several schedules available, the one that has been implemented is the durability schedule for passenger cars. It was chosen to start with the passengers' car schedule since it is the most common and applicable on most of the models produced by the company.

2.1.2. Test tracks

In order to perform the testing schedule, the vehicle undergoes several standardized modules each being a combination of most of the tracks present in the Balocco proving ground.

As a matter of fact, the track's offer of Balocco varies from the low speeds of the 56 special paving or the off-road, to the over 300 km/h possible on the Alfa Romeo track and the High-Speed Track ring, via the 21 km and 147 corners in total on the Langhe track.[1]

Accurately designed to satisfy any testing necessity for passenger's vehicles, sports car and even professional vehicles such as trucks.

Just a part of them is used for the passengers' car's durability schedule

2.1.2.1. Belgian cobblestone (PABA)

The Belgian cobblestone is part of the Alfa special paving as the following three tracks and as the following is a 400 meters long stretch of medium sized bricks perfectly aligned horizontally.

Figure 1 In this picture we can notice from left to right: the short-wave paving, the Procaccini paving, the Swiss Italian pavé, a bumpy asphalt and finally the Belgian cobblestone

Figure 2 View from above of the Alfa special paving tracks

2.1.2.2. Swiss Italian Pavé (PORF) and Procaccini (PROC)

Other two tracks of Alfa special paving are the Swiss Italian pavé and the Procaccini paving they are both stone paving as picture 1 shows and quite similar to one another and to the Belgian cobblestone, from which differ simply in size and relative position of the stones. They are again 400 meters long stretches.

2.1.2.3. Short wave (OB)

Again, among the Alfa special paving is the short wave, that still is a 400 meters long stretch which simulates a portion of road with continuous wavy behavior, hence the name.

All the Alfa special paving tracks are designed to verify the vehicle response and comfort to high frequency vibrations.

2.1.2.4. Langhe (LNG)

The Langhe tracks are a reproduction of a mixed hilly route. It consists of 4 communicating tracks, totaling almost 21 km with 147 corners. The two main tracks are the 7-km Langhe Internal and the 5.3-km Langhe External. The track is designed to analyze the vehicle's reaction to moderately fast turns in the hills.

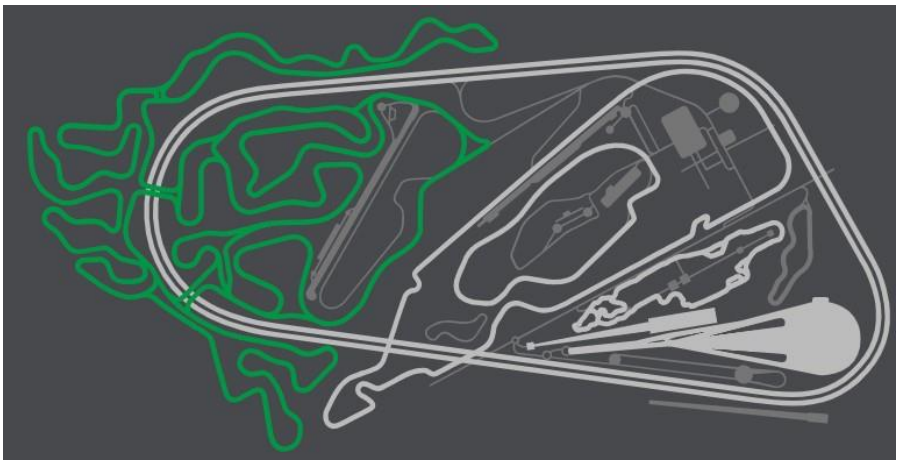


Figure 4 Full Langhe track

Figure 3 Langhe internal aerial view

2.1.2.5. Langhe special paving (PLX)

Langhe Special Paving is a fatigue track to simulate uneven road surfaces and various types of obstacles; it is used for rolling tests related to comfort, suspension and noise; the track has a total length of 3350 meters with 22 turns and 23 obstacles similar to the ones shown in the following figure.

Figure 5 Langhe special paving track

Figure 6 Examples of some of the 23 obstacles present in the PLX track

2.1.2.6. Mixed CISA (MCISA)

The Cisa Track is a fast durability track to simulate a country road with a rough tarmac surface and deep potholes. It is mainly used for comfort, reliability and fatigue testing. The track is 2400 meters long and is narrower than an average road.

Figure 7 A part of CISA track

2.1.2.7. High-Speed (AV)

Figure 8 Aerial view and complete map of the High-Speed track

The High-Speed track simulates 8 km of a fast three-lane highway. Three straights up to 2,000 meters long are connected by three banked curves with a lateral gradient up to 30%, the total length of the track is 7800 meters.

2.1.2.8. Potholes (POT)

The potholes track is a 180 meters long stretch with 4 potholes on it, 2 for each side and it is used for comfort and reliability testing. This is part of Alfa special paving sector.

Figure 9 Potholes

2.1.2.9. High ramps (RAL)

The high ramps are a set of slopes built on an artificial hill with a range of grades and road surfaces varying from a minimum of 5% to a maximum of 30% of incline. They are designed for take-off tests on dry, wet and mu-split surfaces.

Figure 10 High ramps track

2.1.2.10. Symmetric and asymmetric long wave (OL and OL-DS)

The symmetric and asymmetric long wave track has bumps pretty far apart among one another and while for the symmetric part they are in phase meaning the left and right side of the vehicle experience the bump at the same time, in the asymmetric track they are not in phase between the left and the right side. This track tests the durability as always and the steadiness of the vehicle too. As the following two tracks these are part of Alfa special paving too.

2.1.2.11. Humps and slabs (LAS)

The humps and slabs track is an 800 meters long stretch of road with changing surfaces and, as the name suggest, there is a first section with a series of humps and the following parts is made using big slabs of stone.

2.1.2.12. Sleepers and Chrysler potholes (TR-BU)

Sleepers and Chrysler potholes track has been implemented since the formation of the FCA group and consists of steel sleepers and deep potholes distributed along a 400 meters long stretch.

2.1.2.13. Gravel road (BIA)

Figure 11 A turn of the gravel road track

The Gravel Road Track simulates a white road with an even gravel substrate characterized by low grip. Tests of reliability, soiling and handling are carried out on this track. It has a total length of 1.500 meters and 9 turns.

2.1.2.14. Salty ford (GUA)

Figure 12 Salty ford tunnel

The Salty Ford Track is a tunnel with ford designed for corrosion testing.

When entering the track, a sensor activates the nozzles spraying the vehicle with a saltwater solution of NaCl 1.5-2.0 % + CaCl 0.15-0.20 %. The ford is 50 meters long, it has a concrete base and a maximum depth of 15 centimeters.

2.2. Schedule preparation and set up

A priori there is a preparation of the documentation for daily monitoring, testing progress and logbook all of which must be updated on a daily basis mainly by the driver. After the documentation a test on track is done to certify that the vehicle is suitable to start the test in terms of safety and functionality.

2.2.1. Set-up

The first phase of the preparation of the vehicle consists in its set-up. For this reason it's necessary to:

- Measure the wheel pressure and restore design values, if needed;
- If not done previously, run a distance for tire break-in. This operation must be carried out using an high speed track;
- Perform weights measurement according to standards;
- Measure the wheel; restore the right values, if needed;
- Check shock absorbers temperatures: measure the temperatures reached by the shock absorbers in the worst-case ;
- Check that the vehicle is provided with engine speed, otherwise fit one in position readable by the driver. Its fastening must be guaranteed also in case of rollover;
- Equip the vehicle with two-way radio.

2.2.2. Vehicle checks

2.2.2.1. Performance checks

There are 4 performance checks to be executed during an durability schedule at: 0%, 33%, 66% and 100% of the test.

To execute the performance checks, during the test, it's necessary that the components subjected to wear like tire and breaks are in a good state to maintain the performance and ensure the overall significance of the evaluation, hence those are the components subjected to the check.

2.2.2.2. Tightening torques - Works on the vehicle during the test

The works needed are under the responsibility of the Fasteners team, called by the Durability team.

- Each component's disassembly must be anticipated by the verification of the residual torque;
- Each component's assembly must be carried out with its tightening torque certification;
- The decision of the reuse or replacement of the screws it is left at the expertise of the technician that acts on the screw;
- Screws' reusing:
    - Replace the loose fasteners after 2 tightening done applying torque–angle;
    - Replace the loose fasteners after 3 tightening done applying imposed torque.

2.2.2.3. Static inspections

In order to allow an adequate visual inspection and, at the same time, to evaluate the behavior of the vehicle to its washing, it's necessary before each periodic inspection and at the end of the test to:

- Thoroughly wash the vehicle, including the underbody;
- Thoroughly clean the vehicle's interiors

During the test, the following inspections have to be done:

- At the end of each shift;
- At 16%, 33%, 50%, 66% of the test;
- At the end of the test.

In the middle test inspections, parts removal is allowed, for the components removable with no disassembly as the spare tire.

It's possible to disassemble all the necessary components to allow a final inspection at the end of the test. Usually the inspected parts include: vehicle suspension, engine suspension, bumpers, carpets, seats, lights and battery.

2.2.3. Summary of the durability schedule

At the start of the test, it's necessary to execute a list of acceleration's maneuvers to stress the exhaust pipe. After the acceleration cycles, the test continues with the execution of the 5 modules described below:

- «Heavy Roads»: very rough roads with high vertical loads;
- «Hill»: sequence of curves run at medium-high lateral acceleration >= 0,6g; it includes a section with positive and negative obstacles;
- «Suburban»: sequence of curves run at medium lateral acceleration with a lot of gearshifts and accelerations; it includes some lane changes, and it is partially on rough road;

- «Motorway»: medium speed driving with some large radius curves; it includes hard braking in straight lane;
- «Urban »: sequence of positive and negative obstacles run at medium/low speed.

Each of these 5 modules is a combination of one or more of the tracks described in the chapter dedicated to Balocco proving ground.

The schedule designed for passengers' cars condensates the entire life of the vehicle through a test long almost 24 weeks [4].

Load conditions of the vehicle during the test execution change and are the following

- 66% heavy load;
- 34% light load.

Where heavy load is the condition which emulates the maximum load that the vehicle is supposed to carry, and "light load" is the condition with the least weight possible.

2.2.4. Durability schedule's modules details [5]

2.2.4.1. Acceleration module at start of test

Before the start of the test schedule, execute the acceleration modules, some cycles in light load and some cycles in heavy load.

Each of these cycles consists of accelerations test from both null velocity and while rolling. Moreover, brisk accelerations are performed on passengers' cars too to check the clutch resistance.

2.2.4.2. HEAVY ROADS

Proceed on "Alfa special paving" and perform the following events sequence.

The complete sequence constitutes 1 repetition of the following module keeping a constant low speed:

1. Belgian cobblestone
2. Short wave and Procaccini
3. Belgian cobblestone
4. Italian-Swiss pavé and Procaccini
5. Belgian cobblestone
6. Italian-Swiss pavé and Procaccini
7. Belgian cobblestone
8. Italian-Swiss pavé and Procaccini

2.2.4.3. HILL

Afterwards, perform the below sequence with sport driving to carry out one repetition of the module.

1. Langhe Internal (first part)
2. Langhe External (first part)
3. Langhe Special Paving:
4. Langhe external (second part)
5. Langhe internal (second part)

2.2.4.4. SUBURBAN

Proceed on "Mistino endurance" track:

- one lap with partial CISA so only the straight line part and the entrance on the CISA has to be with speed <= 10 Km/h.

- Direction: counterclockwise;
- Driving style: sporting pace with a target for the lap time, to be defined for every vehicle before the test start.

2.2.4.5. URBAN

Proceed on “Alfa Special Paving” and perform the below sequence to carry out one repetition of the module.

1. Symmetric long wave and Humps
2. Slabs
3. Asymmetric long wave Right and Left
4. Symmetric long wave and Humps
5. Smooth asphalt
6. Sleepers
7. Chrysler potholes and smooth asphalt
8. Potholes
9. High Ramps

2.2.4.6. MOTORWAY

Proceed on “High Speed track” and perform the below sequence to carry out one repetition of the module.

- High Speed: every Lap: 3 braking from 130 Km/h with ABS activation.

3. CRITERIA OF EVALUATION OF THE TESTS

Once the track tests have been performed all the data is collected, filtered from measuring and signal errors and stored.

Afterwards, the analysis starts and in the case of durability testing, there are three main parameters that are checked in the collected signals: symmetry, average damage and execution of the test. To achieve this goal the software mainly used in the company to carry out fatigue analysis is nCode by HBK.

The nCode software provides a powerful range of solutions to process measured data, perform durability analysis and FE-based fatigue analysis, find insights, and manage data.

Three tests for each track must be provided by the Balocco Proving Ground team and among these three, one must be selected as Golden Pass according to the three parameters which are exposed next.

3.1. Symmetry

The symmetry analysis has the objective of evaluating if the left side and the right side of the car react in a similar way to the excitations that the vehicle undergoes.

In 2024 Stellantis still verifies the symmetry parameter by manually checking two level crossing graphs generated by nCode program; the graphs analyzed correspond to symmetric accelerometers installed on the vehicle.

If the graphs almost overlap in the sense that have similar maxima and minima and the levels have a similar trend towards the end the two analyzed channels of the acquisition pass the symmetry test.

Clearly this modus operandi is extremely time consuming since on a single vehicle are normally installed tens and more recently hundreds of accelerometers corresponding each to a channel of the data.

An example of two symmetric channels' level crossing graph is shown.

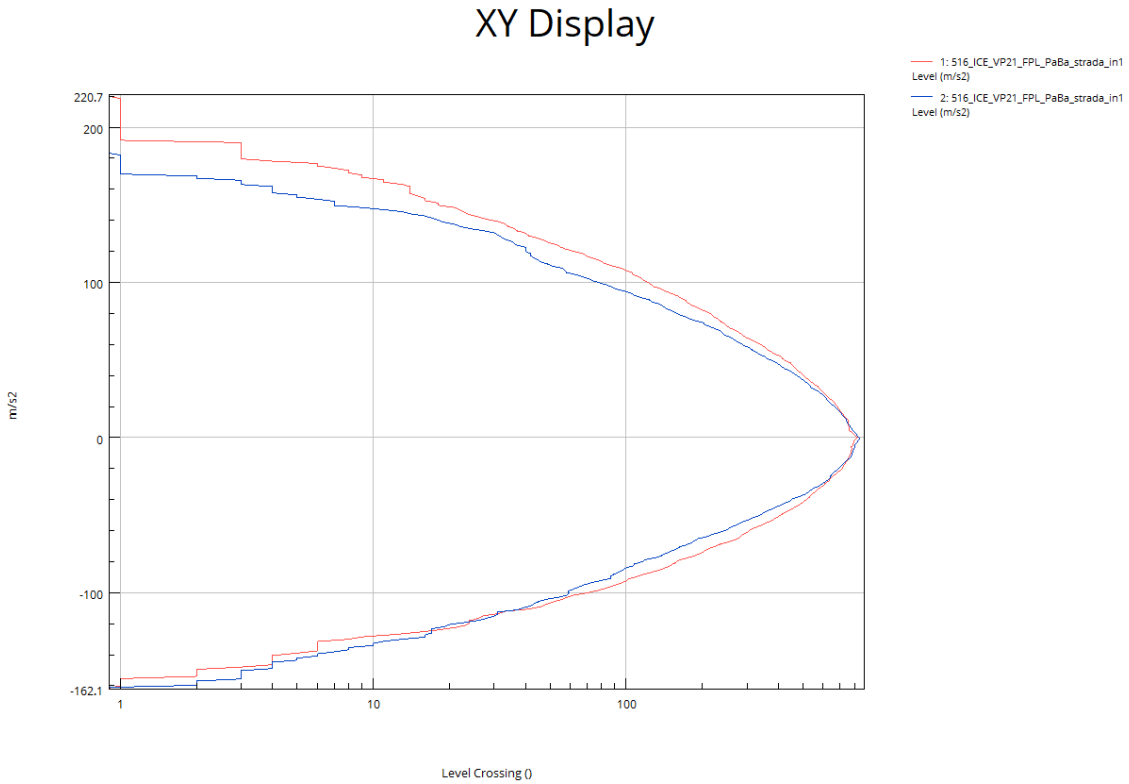


Figure 13 Level crossing graph generated on nCode software

When all the symmetric channels have been analyzed, the operation is repeated for the three best test data and the one among the three which has higher symmetry and less errors, is consider the best try.

3.2. Average damage

The average damage analysis is performed by analyzing the cumulated damage of a cycle according to each accelerometer on the car.

The goal of these analysis is not to determine which among three tests is the less damaging but is actually to determine the middle damaging one so that the prediction is neither underestimated nor overestimated.

This test was already implemented in an excel file as the one shown.

	P1	Root[(D/A-1)^2] Channel Count	0,229	P2	Root[(D/A-1)^2] Channel Count	0,234	P3	Root[(D/A-1)^2] Channel Count	0,326	Pass Variation	0,05
		D/A	1,01		D/A	0,95		D/A	1,04		
		Rank	1		Rank	2		Rank	3		

Channel Name	#	Damage	Damage / Average Dmg	$(D/A-1)^2$	Damage	Damage / Average Dmg	$(D/A-1)^2$	Damage	Damage / Average Dmg	$(D/A-1)^2$	Average Damage	$\sum(D/A-1)^2$
LtFr Spindle Load (X)	1	2,96E+10	1,0186	0,0003	2,34E+10	0,8044	0,0383	3,42E+10	1,1770	0,0313	2,91E+10	0,0699
LtFr Spindle Load (Y)	2	5,15E+09	1,1091	0,0119	5,03E+09	1,0816	0,0067	3,76E+09	0,8093	0,0364	4,65E+09	0,0549
LtFr Spindle Load (Z)	3	5,35E+11	0,5987	0,1611	1,06E+12	1,1867	0,0348	1,09E+12	1,2147	0,0461	8,93E+11	0,2420
LtFr Spindle Moment (X)	4	2,84E+07	1,2362	0,0558	1,74E+07	0,7600	0,0576	2,30E+07	1,0038	0,0000	2,29E+07	0,1134
LtFr Spindle Moment (Y)	5	1,20E+06	1,3542	0,1255	7,22E+05	0,8180	0,0331	7,31E+05	0,8278	0,0297	8,83E+05	0,1882
LtFr Spindle Moment (Z)	6	6,12E+05	0,8489	0,0228	9,21E+05	1,2791	0,0779	6,28E+05	0,8720	0,0164	7,20E+05	0,1171
RtFr Spindle Load (X)	9	3,08E+10	1,1416	0,0200	2,70E+10	0,9999	0,0000	2,31E+10	0,8585	0,0200	2,70E+10	0,0401
RtFr Spindle Load (Y)	10	4,14E+09	0,8179	0,0332	6,19E+09	1,2229	0,0497	4,85E+09	0,9592	0,0017	5,06E+09	0,0845
RtFr Spindle Load (Z)	11	1,33E+12	1,1359	0,0185	1,24E+12	1,0630	0,0040	9,37E+11	0,8011	0,0396	1,17E+12	0,0620
RtFr Spindle Moment (X)	12	1,39E+07	0,9385	0,0038	1,68E+07	1,1370	0,0188	1,37E+07	0,9245	0,0057	1,48E+07	0,0282
RtFr Spindle Moment (Y)	13	1,19E+06	1,3445	0,1187	6,63E+05	0,7459	0,0646	8,08E+05	0,9096	0,0082	8,88E+05	0,1915
RtFr Spindle Moment (Z)	14	9,93E+05	0,8360	0,0269	1,78E+06	1,4963	0,2464	7,93E+05	0,6677	0,1104	1,19E+06	0,3837

Figure 14 Excel example of average damage analysis with the three tries

To better explain the average damage procedure, a legend and a zoomed picture of the excel file is shown.

Legend of the terms as written in the excel picture:

- Damage: is a value directly calculated from the accelerometer readings through the SN fatigue analysis block of nCode
- Average damage: is the average between the three damages of the same channel for the different tests
- Damage / Average dmg: to understand how far each test is from the average (if 1 they are equal)
- $(D/A-1)^2$: is the squared deviation of "Damage / Average dmg" from the ideal value 1
- Root[$(D/A-1)^2$]: is the standard deviation of "Damage / Average dmg" calculated for each test considering all channels
- D/A: is the mean of "Damage / Average dmg" calculated for each test among all channels

	P1	Root[(D/A-1)^2] Channel Count	0,229	Pass Variation:	0,05
		D/A	1,01		
		Rank	1		

Channel Name	#	Damage	Damage / Average Dmg	$(D/A-1)^2$	Average Damage	$\sum(D/A-1)^2$
LtFr Spindle Load (X)	1	2,96E+10	1,0186	0,0003	2,91E+10	0,0699
LtFr Spindle Load (Y)	2	5,15E+09	1,1091	0,0119	4,65E+09	0,0549
LtFr Spindle Load (Z)	3	5,35E+11	0,5987	0,1611	8,93E+11	0,2420
LtFr Spindle Moment (X)	4	2,84E+07	1,2362	0,0558	2,29E+07	0,1134
LtFr Spindle Moment (Y)	5	1,20E+06	1,3542	0,1255	8,83E+05	0,1882
LtFr Spindle Moment (Z)	6	6,12E+05	0,8489	0,0228	7,20E+05	0,1171
RtFr Spindle Load (X)	9	3,08E+10	1,1416	0,0200	2,70E+10	0,0401
RtFr Spindle Load (Y)	10	4,14E+09	0,8179	0,0332	5,06E+09	0,0845
RtFr Spindle Load (Z)	11	1,33E+12	1,1359	0,0185	1,17E+12	0,0620
RtFr Spindle Moment (X)	12	1,39E+07	0,9385	0,0038	1,48E+07	0,0282
RtFr Spindle Moment (Y)	13	1,19E+06	1,3445	0,1187	8,88E+05	0,1915
RtFr Spindle Moment (Z)	14	9,93E+05	0,8360	0,0269	1,19E+06	0,3837

Figure 15 Detail of one test for average damage analysis

The analysis I performed in the following way:

- Consider three tests for each track

- Weight the different channels based on the type of track and the type of axis measured
- For each channel calculate the average of the three test and use it to standardize the damage
- Evaluate the standard deviation of the damage of each channel from the average
- Extrapolate the weighted mean of D/A (damage/average) and the weighted mean's root of the standard deviation
- Finally choose the best test based on the mean damage and/or the lower standard deviation values depending on the data's accuracy and nominal values.

3.3. Execution of the test

This third parameter is an analysis that does not focus on the accelerometers data, instead it consists in verifying whether or not the vehicle has performed the test according to the forementioned standards.

This analysis is carried out visually too and an operator has to check the vehicle's speed time history during the test.

Examples of the speed graphs are shown.

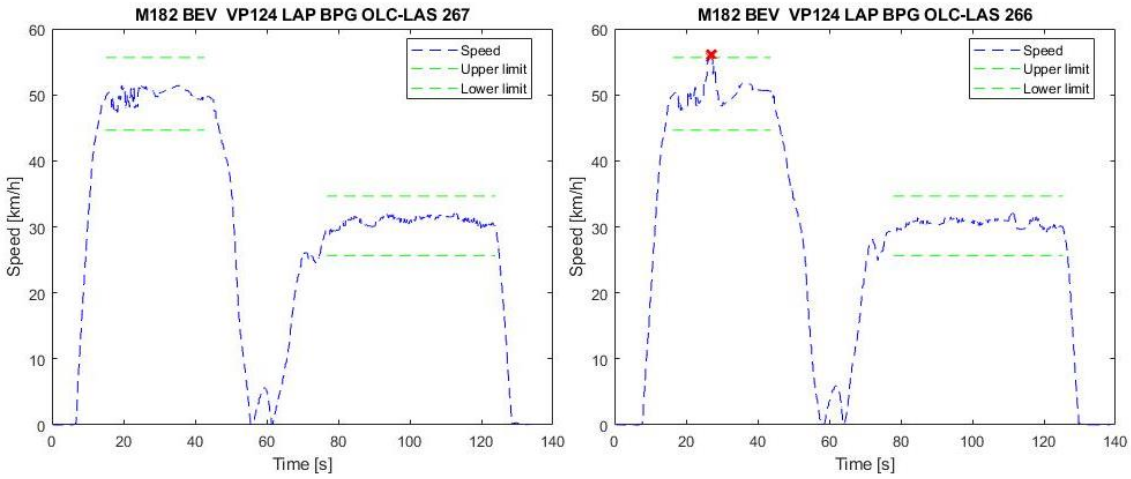


Figure 16 Speed graphs of two tests on the same track

In the graphs of the same track, it can be seen that there is the test on the left which perfectly respects the speed limits while that on the right has some points of overspeed. Surely considering the better steadiness of the speed, the test on the left, in this comparison is to be preferred to the one on the right.

4. PARAMETRIZATION OF THE CRITERIA

The aim of the thesis is to make the durability analysis totally independent from a human operator but should still leave the possibility for an operator to check on the performed analysis and the results obtained. As a matter of fact, it is important to check

if the “Golden Pass” test selected by the MATLAB app upon the three valid test is actually accurate or not.

Therefore, instead of revolutionizing the testing, it was simply parametrized in such a way that a computational analysis could be carried out in a way similar to the traditional durability analysis so that it is not required a strong staff formation to integrate the MATLAB application in the present durability analysis structure.

In order to give though a definite and objective result to the analysis, the parametrization aims to determine through numbers and mathematical instruments which is the best test out of the three analyzed for each track. To achieve this goal each test will have three marks, one for symmetry, one for average damage and one for execution respectively.

The marks will range from 1 to 10 and the test with the best weighted average will be the chosen Golden Pass test for that track.

4.1. Parametrization of symmetry

The symmetry analysis, among the three, is definitely the most complex one because the traditional testing was carried out visually on graphs, so it requires quite the effort to reproduce a visual graph analysis numerically.

For starter, the symmetry analysis relies on the level crossing outputs of each couple of symmetric channels. The level crossing analysis output is a two-column table representing the value of acceleration assigned to a specific level crossing interval and the number of times the examined part has experienced said acceleration. While the number of level crossing intervals is assigned by the standard as 256, the values of each interval are extrapolated from the time history of the accelerometer in exam and can vary greatly from one another.

A part of a level crossing output file is shown in the picture below.

LF_AccZ_Spindle	
Value	Rep
-12.0901	0
-11.9864	2
-11.8828	3
-0.792353	835
-0.688704	960
-0.585055	1081
-0.17046	1943
-0.0668116	2411
0.0368372	3056
0.140486	2268
0.244135	1832
0.347783	1570
0.65873	1045
0.762379	922
14.1331	1
14.2367	1
14.3404	0

Figure 17 Level Crossing output table example

Cleary in the figure shows the most characteristic parts of the level crossing table hence the minimum value at the top, the maximum value at the bottom and the most common value at the center which has the maximum number of repetitions.

There is one table for each channel and obviously, as mentioned, two symmetric channels at the time are considered.

If the level crossing tables were directly used, these would result into sparse points on an Acceleration-Repetitions graph which are a hard instrument to use in an analysis, especially since it cannot be given for granted that the intervals have the same values even if the range should be similar for a symmetric couple of channels.

According to this argument, it was chosen to approximate each channel' sparse points into curves that result in an easier manipulation of the data by setting the same precise points to be analyzed on each approximated curve.

4.1.1. Data approximation

4.1.1.1. Polynomial approximation

The polynomial approximation was the first one to be considered, it was performed by splitting the data into upper and lower part so to keep the same graph format traditionally used in the company where the acceleration is graphed on the y axis and the repetitions on the x-axis. Despite being extremely simple at low polynomial grades, at first it gave precise enough results, but as the complexity of data increased the polynomial grade needed to augment accordingly and it was not an ideal solution since the polynomial grade would need to change for each couple of symmetric channels. It was tried to set a reference polynomial grade that could fit well any channel but, in some cases, quite strong oscillations emerged in the graphs, particularly in the middle region where the repetitions are high. A significant case of oscillations in the level crossing approximation graph is shown in the picture below.

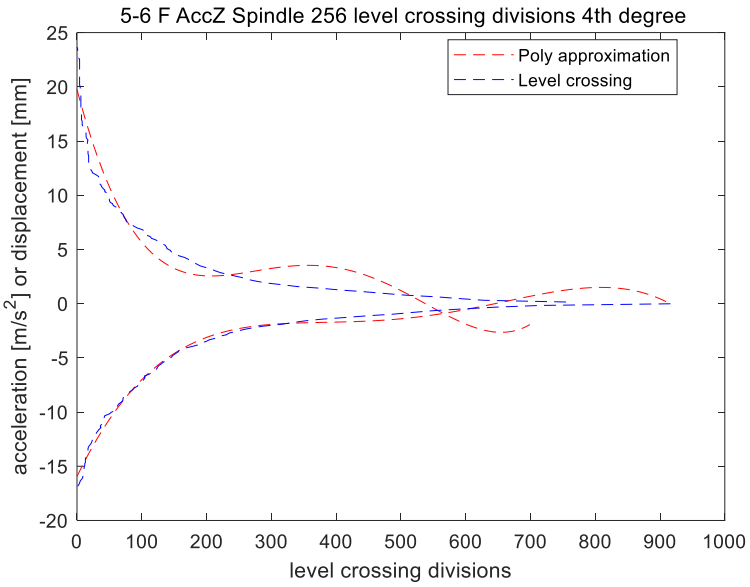


Figure 18 Polynomial approximation with respect to level crossing points curves

Since a low degree polynomial approximation would not be thoroughly precise for all the graph, it was chosen an alternative solution.

4.1.1.2. Spline interpolation

Instead of polynomial approximation, it was chosen to start the process with a spline interpolation of the level crossing points which, despite being more computationally demanding, since it is a piecewise polynomial function, it gives the best result for sparse points data as it is required that the spline passes through all the points without any oscillations in between.

A further modification to better approximate and represent the level crossing data is to analyze the data in a graph with the level crossing interval values on the x-axis and the number of repetitions on the y axis so that the representation is a Gaussian like continuous curve that does not require to be split between upper and lower part. The graph represented in this way deviates from the traditional one analyzed in the company but is way more efficient for the computational point of view and since the difference is mainly graphical, it was chosen to proceed with the latter option.

In the following pictures is shown the difference between the polynomial approximation and the one using the spline function comparing the same couple of symmetric channels.

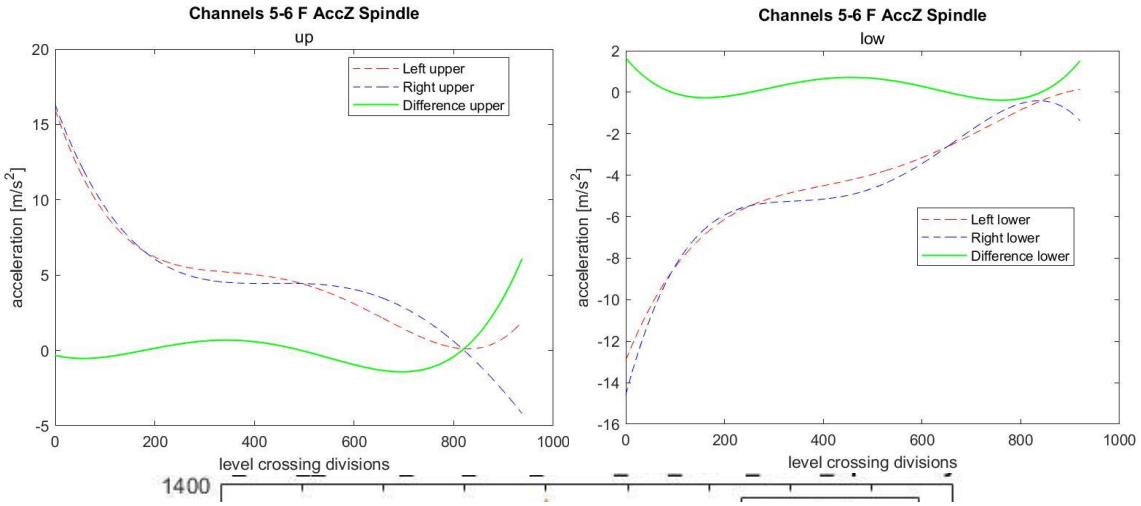


Figure 20 Graph using direct polynomial approximation

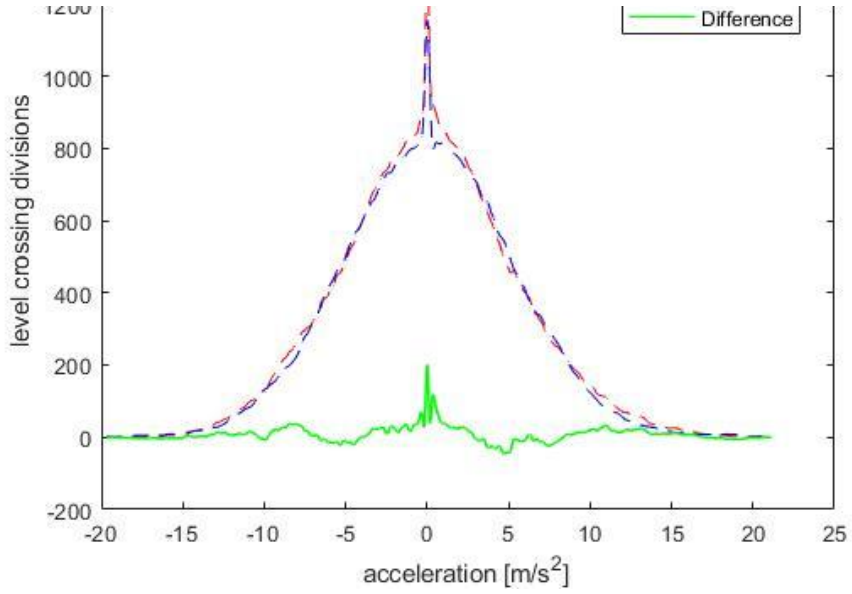


Figure 19 Graph using spline approximation

4.1.2. Parameters for best test choice in the symmetry analysis

To choose the mark for the symmetry analysis, there are four parameters that were considered for each couples of symmetric channels.

- Standard deviation of the difference signal which accounts for 45% of the final mark
- Difference among maxima which accounts for 7.5% of the final mark
- Difference among minima which accounts for 7.5% of the final mark
- Integral of the difference signal which accounts for 40% of the final mark

4.1.2.1. Standard deviation

The difference signal is obtained by subtracting the level crossings approximations of the two symmetric channels analyzed. If the two level crossing approximations were identical as it would be ideal, the difference would be a straight line of zeroes hence the x-axis. Instead, this is clearly never the case and the signal will oscillate around zero.

Therefore the standard deviation of the difference signal is calculated and the lowest the value of the standard deviation the more symmetric are the accelerations experienced by a component, hence it has the highest mark.

4.1.2.2. Difference among maxima

Besides the difference of the overall level crossing approximation is crucial to check if the extremes of the acceleration experienced by the vehicle's component are similar in the left and the right part.

To implement this parameter the difference between the maxima value of the level crossing interval of the two symmetric channels is calculated.

Also in this case, the lowest difference value corresponds to the most symmetric test hence having the highest mark.

4.1.2.3. Difference among minima

According to the previous paragraph as the minimum value is also an extreme of the level crossing analysis, the difference between the two minima values is calculated too.

The same rule that was true for maxima applies and the lowest difference value corresponds to the most symmetric test hence having the highest mark.

4.1.2.4. Integral

Finally, the last parameter of the symmetry analysis is the cumulated trapezoidal integral of the difference function among the two level crossing curves.

Instead of a simple trapezoidal integral that would consider the area of the curve below the x-axis as negative, in this case the area portions below each interval of two subsequent point are summed among one another in absolute value hence giving an idea of how close from the x-axis the difference signal actually is.

Like the other three parameters, the lowest integral value corresponds to the most symmetric test hence having the highest mark.

4.2. Parametrization of average damage

The average damage analysis is the simplest to implement since it already has mathematical foundations to be evaluated.

As described the damage is calculated directly through a function of the nCode software, once the damage corresponding to each channel is given as output of the SN fatigue function of nCode, the following steps are performed for the three test of each track:

- For each channel calculate the average of the three test and use it to standardize the damage value
- Evaluate the standard deviation of the damage of each channel from the average
- Extrapolate the weighted mean of D/A (damage/average damage) and the weighted mean's root of the standard deviations

4.2.1. Parameters for best test choice in the average damage analysis

The average damage grading is fairly similar to the symmetry one, but the characteristics evaluated for each channel among the three tries for each test track in this case are:

- Damage/Average damage value which accounts for 55% of the final mark
- Standard deviation which accounts for 45% of the final mark

4.3. Parametrization of execution

The execution analysis, at first sight, has a straightforward implementation too since there are simply some speed ranges, breakings or accelerations that the vehicle has to follow for each track.

For instance, on the special pavings, the speed has to be low and the focus is on the comfort, on the opposite with respect to the high way ring where the performances of the car are put to the test, therefore concentrating on speed and breaking.

In order to check these rules there is no approximation needed like there was in the symmetry analysis since every car performing the schedule has also GPS on it which collects and store the following data:

- Speed
- Latitude
- Longitude

Along with these three signal collected from the GPS, is also required the data channel corresponding to the pressure applied on the brake pedal to check if there was enough pressure to activate the ABS during the breaking.

Once this data is collected the upper and lower limits of the speed signal are set for each track and the following parameters are considered.

4.3.1. Parameters for best test choice in the execution analysis

4.3.1.1. In or out verification

This is an "a priori" check where if there are too many data points of the speed range established from the standard, there is a big penalty in the mark of this test.

A punctual verification is performed and it is either given a full mark if there are not any points out of boundary or a reduced one if there are some, does not even matter how close the speed is to the limit one, for this parameter, as far as it is inside the range.

4.3.1.2. Standard deviation

This second parameter instead, checks how close the speed is to the ideal one dictated by the standards, but since it cannot be asked an extreme precision from the test driver considering the play of the accelerator and a natural delay of the speed correction procedure, this parameter has a lower weight on the execution mark.

5. nCODE HANDLING OF TIMESERIES

When all the data of the durability schedule have been collected and stored, they all pass through the nCode software which is used by the Stellantis group to handle the accelerometers signals from the test track.

Using nCode software the signals are filtered, truncated to the parts of interest and eventually if there are no major errors present, they pass to the durability analysis section.

Since nCode is the most used program for handling signals, it was used to directly manipulate the accelerometers' timeseries. It has been written an nCode flow which imports the timeseries of the tests of a vehicle and for each channel:

- Calculates the level crossing table for the symmetry analysis;
- Calculates the damage according to the SN fatigue analysis;
- Extracts the GPS data needed for the execution analysis.

Finally, the flow exports the files in csv format in three different folders inside the vehicle testing folder; each of this folder corresponds to a specific analysis: symmetry, average damage or execution.

6. MATLAB APPLICATION

This paragraph means to start the analysis of the MATLAB coding portion of the thesis.

Since the app is composed by four different MATLAB functions and the application itself which can be programmed either in "Design view" where the user interface can be modified directly and the "Code view" where it is necessary to write code lines.

It was chosen to start analyzing the MATLAB portion form the application part and gradually move inside the single function.

6.1. User interface

The user interface of the application is really simple and intuitive. It is split into two sections, the first dedicated for the user commands and the second for the reading of the results and occasionally selection of options.

The user interface normally presents itself in three main modes:

- "Ready" the application is waiting for an input folder to be processed;
- "Loading" the application is performing the analysis;
- "Done" the application analysis ended and the results are shown.

Then there are two rarer display modes which are:

- "Error" if a predicted error occurs, the application shows the problem to the user;
- "Channel selection" if the channel selection is set to the manual mode, a selectable list appears.

6.1.1. Ready

In this mode we can clearly see that only the left panel is enabled since the analysis has yet to start. Before clicking the start button, the user can choose which of the three analysis wants to perform and what weight each of the three analyses should have on the average mark of each test.

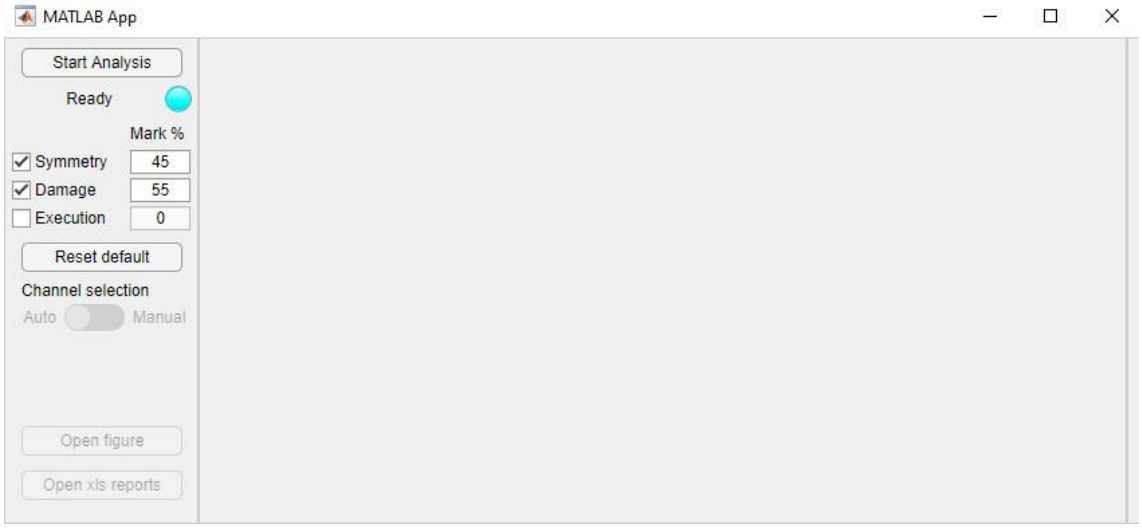


Figure 21 User interface "Ready"

The default option for the analysis is 45% symmetry and 55% damage and in case of subsequent analysis with different marking weights, it can be restored by means of the "Reset Default" button.

6.1.2. Loading

Once the "Start Analysis" button is pressed, the application passes to loading mode.

In this mode all the left buttons freeze and are unusable, a pop-up window appears asking to select the vehicle's folder to be analyzed.

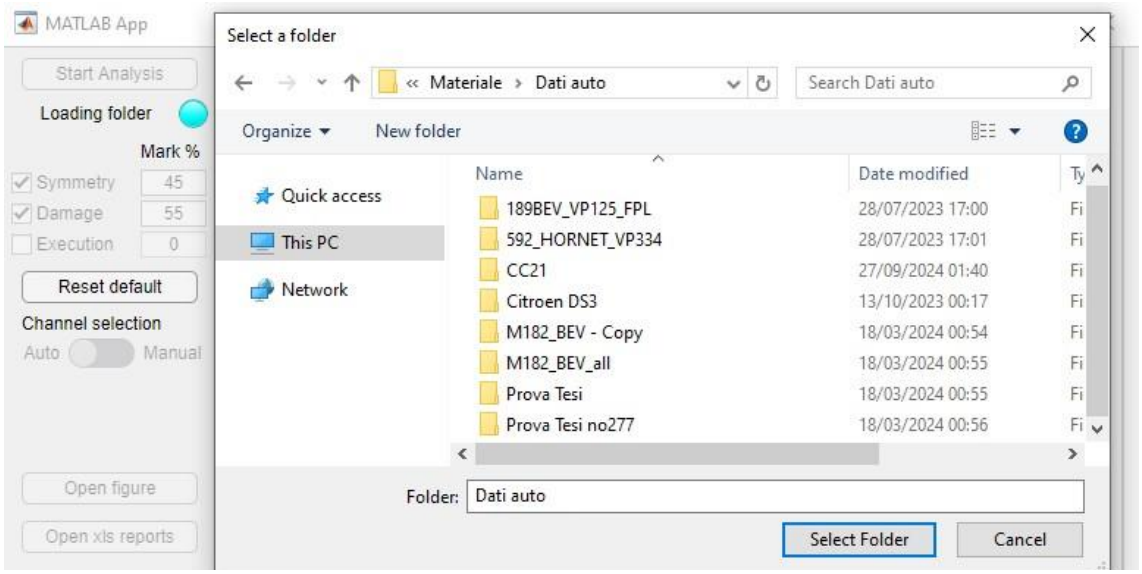


Figure 22 Folder selection

After the folder has been selected there is just a string with a percentage that shows the progress of the analysis and a text that specifies which analysis is running.

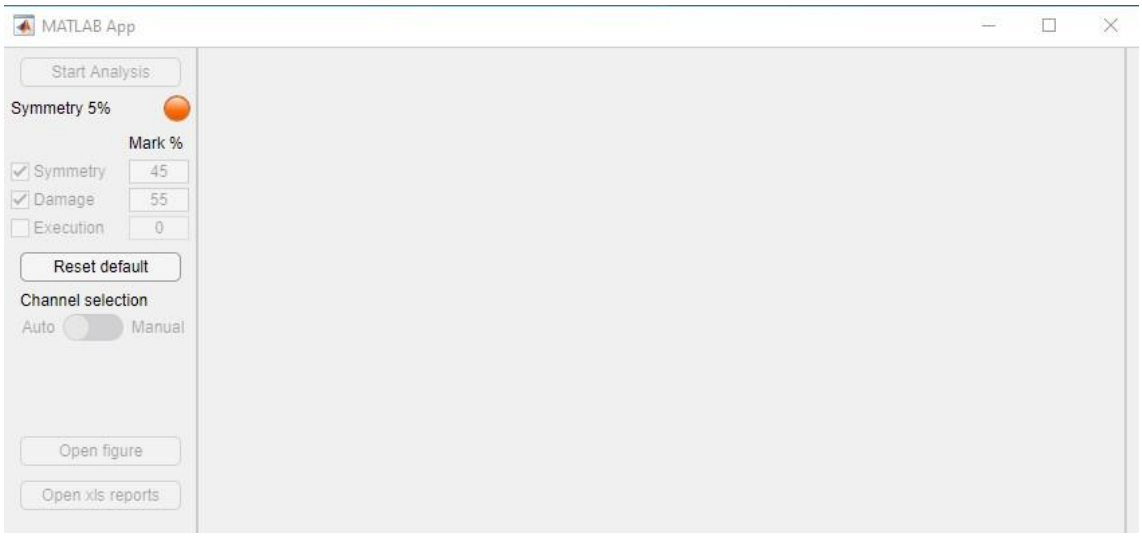
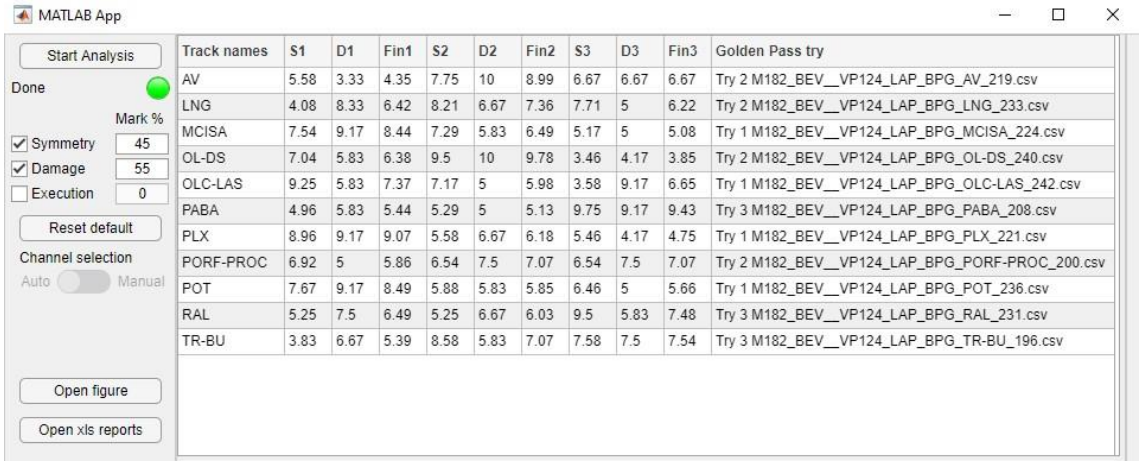


Figure 23 User interface "Loading"

6.1.3. Done

As soon as the analysis ends the application switches to the done mode.



Track names	S1	D1	Fin1	S2	D2	Fin2	S3	D3	Fin3	Golden Pass try
AV	5.58	3.33	4.35	7.75	10	8.99	6.67	6.67	6.67	Try 2 M182_BEV__VP124_LAP_BPG_AV_219.csv
LNG	4.08	8.33	6.42	8.21	6.67	7.36	7.71	5	6.22	Try 2 M182_BEV__VP124_LAP_BPG_LNG_233.csv
MCISA	7.54	9.17	8.44	7.29	5.83	6.49	5.17	5	5.08	Try 1 M182_BEV__VP124_LAP_BPG_MCISA_224.csv
OL-DS	7.04	5.83	6.38	9.5	10	9.78	3.46	4.17	3.85	Try 2 M182_BEV__VP124_LAP_BPG_OL-DS_240.csv
OLC-LAS	9.25	5.83	7.37	7.17	5	5.98	3.58	9.17	6.65	Try 1 M182_BEV__VP124_LAP_BPG_OLC-LAS_242.csv
PABA	4.96	5.83	5.44	5.29	5	5.13	9.75	9.17	9.43	Try 3 M182_BEV__VP124_LAP_BPG_PABA_208.csv
PLX	8.96	9.17	9.07	5.58	6.67	6.18	5.46	4.17	4.75	Try 1 M182_BEV__VP124_LAP_BPG_PLX_221.csv
PORF-PROC	6.92	5	5.86	6.54	7.5	7.07	6.54	7.5	7.07	Try 2 M182_BEV__VP124_LAP_BPG_PORF-PROC_200.csv
POT	7.67	9.17	8.49	5.88	5.83	5.85	6.46	5	5.66	Try 1 M182_BEV__VP124_LAP_BPG_POT_236.csv
RAL	5.25	7.5	6.49	5.25	6.67	6.03	9.5	5.83	7.48	Try 3 M182_BEV__VP124_LAP_BPG_RAL_231.csv
TR-BU	3.83	6.67	5.39	8.58	5.83	7.07	7.58	7.5	7.54	Try 3 M182_BEV__VP124_LAP_BPG_TR-BU_196.csv

Figure 24 User interface "Done"

It can be seen in this case that the summary table with the results immediately appears filled with the marks for each analysis of each track and with the weighted average according to the selected values.

S stands for symmetry

D stands for damage

E stands for execution

Fin stands for final mark

The column Golden pass indicates the test for each track that has scored the maximum mark.

In this mode all the buttons on the left are enabled once again so if another analysis is to be started it can be started.

Furthermore, there are two buttons at the bottom of the left panel which activate once the analysis is done.

6.1.3.1. Open figure

This button opens a pop-up window directly to the folder in which all the graphs inherent to the last analysis performed are stored.

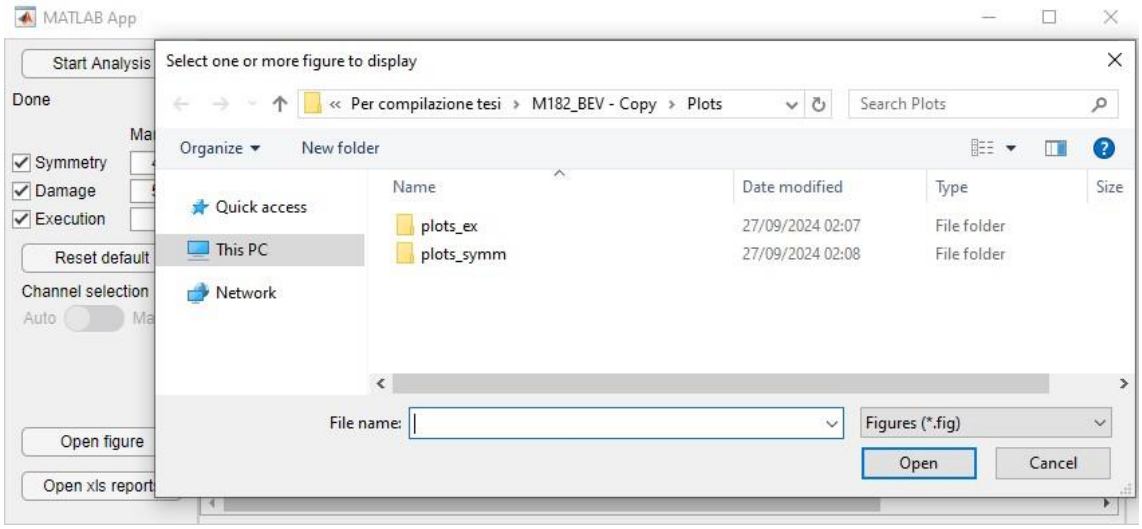


Figure 25 Open figure window

The graphs are stored for the symmetry analysis and for the execution analysis, they are stored in both ".fig" format, which is useful if the graph has to be analyzed using MATLAB, and ".jpg" format useful for a quick share, or to be attached in company' presentations where it is not needed a high resolution.

6.1.3.2. Open xls reports

The open xls reports button opens a pop-up window directly into the folder where the summary reports for each analysis and for the overall Golden Pass selection are stored.

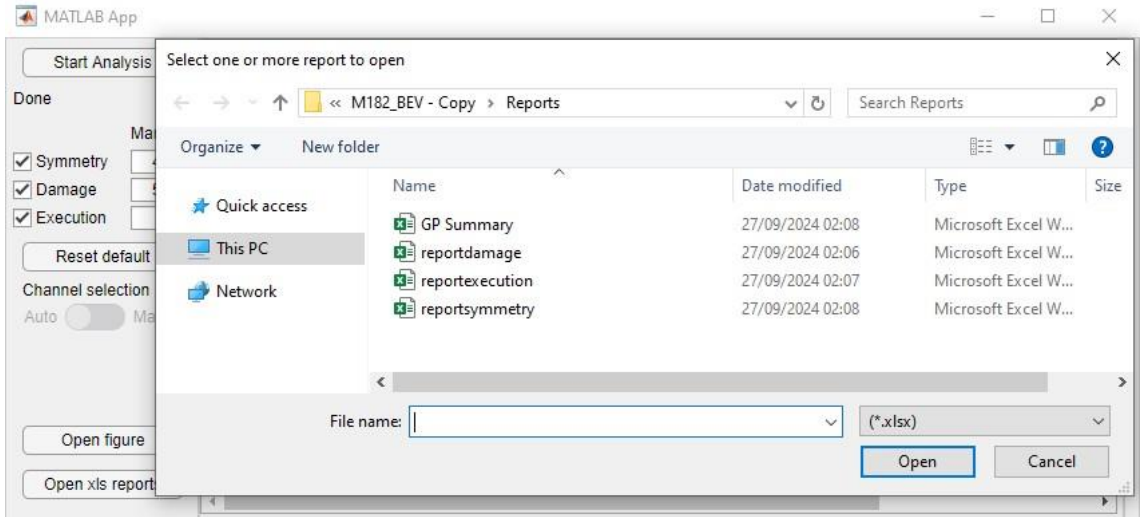


Figure 26 Open xls reports window

They are saved as ".xls" files since they are mostly tables. In this way the operator, should it notice something strange in the summary table in-app, may promptly access the partial reports of each analysis and check the problem.

6.1.4. Secondary displays

6.1.4.1. Error

It appears if an error in the loading or the computing process occurred; obviously it only supports errors introduced in the code, an example is shown below.

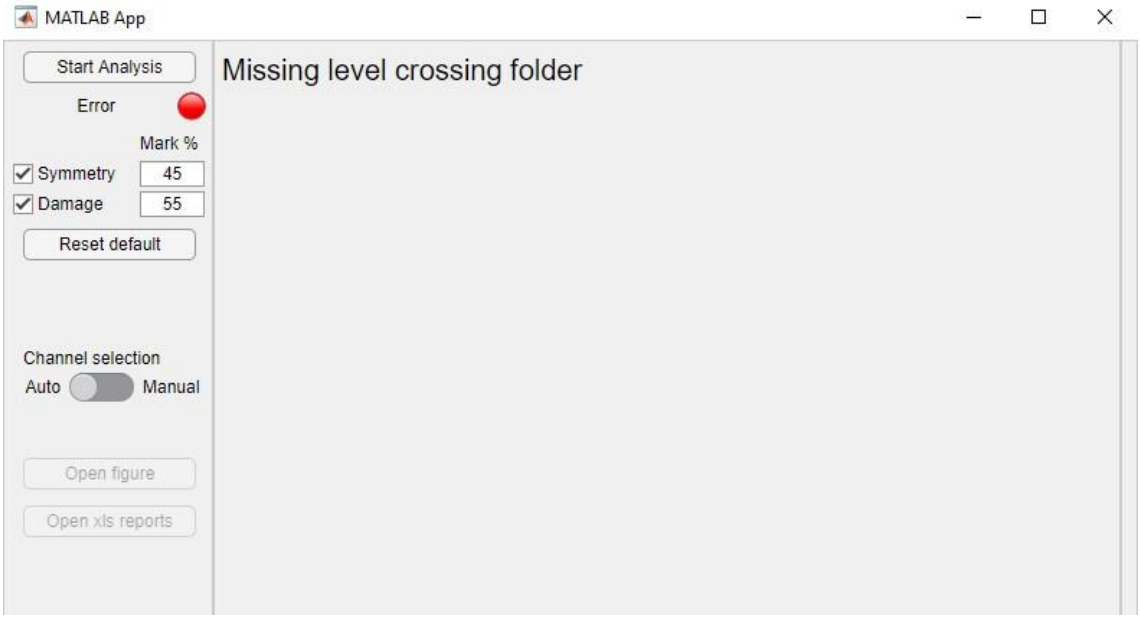


Figure 27 User interface "Error"

In this particular case the problem is fairly simple, so a written message appears and the application does not close since is a minor error.

6.1.4.2. Channel selection

Should the operator decide that the standards channels which are automatically checked during the analysis are not enough or are too many, the toggle "channel selection" can be switched to "manual" in the "Ready" mode and the list of channels present in the selected folder will appear. The user can select all the channels it wants to analyze.

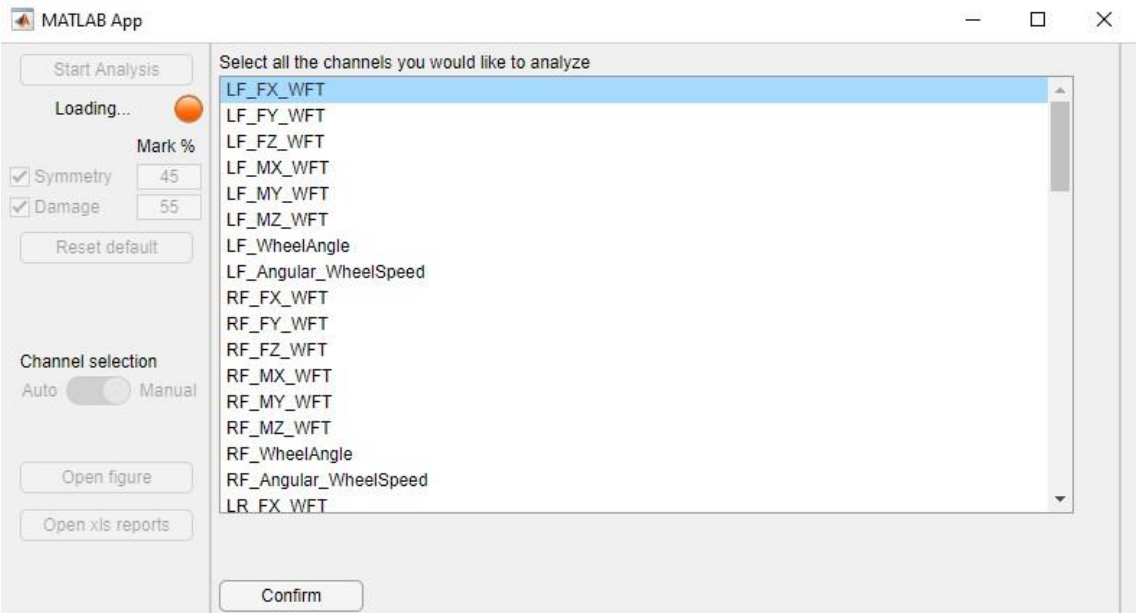


Figure 28 User interface "Channel selection"

Once the user confirms the analysis continues in the regular way checking only the selected channels.

6.2. Code view of the application

Even if MATLAB App Designer has a user-friendly drag and drop design view of the application, everything is written in the source code of the application accessible through the "Code view".

```
271         % Create StartAnalysisButton
272 -       app.StartAnalysisButton = uibutton(app.LeftPanel, 'push');
273 -       app.StartAnalysisButton.ButtonPushedFcn = createCallbackFcn(app, @StartAnalysisButtonPushed, true);
274 -       app.StartAnalysisButton.Position = [13 333 120 22];
275 -       app.StartAnalysisButton.Text = 'Start Analysis';
```

Figure 29 Start analysis button creation code

In the code view all the single elements of the user interface are created with specifics of position, dimension and whether they have to visible and interactive or not; just like shown in the above picture. A lot of other settable options are hidden since are set to their default value.

Besides the elements in the code view is possible also to assign complex behaviors to the push of a button.

6.2.1. Start analysis button routine

The main button concerning the thesis is the “Start Analysis” button.

```
51          % Button pushed function: StartAnalysisButton
52          function StartAnalysisButtonPushed(app, event)
53              if (app.ex_mark_box.Value+app.dam_mark_box.Value+app.symm_mark_box.Value)==100
54                  app.TextArea.Visible = 'off';
55                  app.UITable.Visible = 'off';
56                  app.symm_mark_box.Enable = 'off';
57                  app.dam_mark_box.Enable = 'off';
58                  app.ex_mark_box.Enable = 'off';
59                  app.symm_cb.Enable = 'off';
60                  app.dam_cb.Enable = 'off';
61                  app.ex_cb.Enable = 'off';
62                  app.StartAnalysisButton.Enable = 'off';
63                  app.OpenfigureButton.Enable = 'off';
64                  app.OpenxlsreportsButton.Enable = 'off';
65                  drawnow;
66                  [summary, error, reports_dir, graphs_dir]=main_GP(app); %#ok<*ADPROPLC>
67                  app.reports_dir=reports_dir;
68                  app.graphs_dir=graphs_dir;
69                  app.symm_mark_box.Enable = 'on';
70                  app.dam_mark_box.Enable = 'on';
71                  app.ex_mark_box.Enable = 'on';
72                  app.symm_cb.Enable = 'on';
73                  app.dam_cb.Enable = 'on';
74                  app.ex_cb.Enable = 'on';
75                  app.StartAnalysisButton.Enable = 'on';
```

Figure 30 Start analysis button routine

Following the code in the picture from line 52 it can be seen that once the “Start analysis button” is pushed, there is immediately an if condition in line 53 checking that the sum of the analysis weights for the final mark is 100, if that is not true, the values have to be adjusted.

Afterwards, from line 54 to line 65 all the buttons interactive elements are disabled and in line 66 the function “main” is called where the analysis is performed. The function returns:

- The summary table for the Golden Pass choice;
- The error code to check if an error occurred;
- The graphs and reports’ directories paths.

Once the analysis ends all the elements are enabled again together with the switches for figures and reports.

Instead of analyzing all the other elements which have fairly simple codes, the analysis functions will be analyzed.

7. MATLAB FUNCTIONS

The MATLAB functions on which the application relies are hereafter illustrated. It will be presented following the flow of the application once the “Start analysis” button is pressed.

7.1. Main

The function “main” is the first one called by the application and, as the name says, is the linking function between the three singular analysis.

```
1   function [summary, error, reports_dir, graphs_dir]=main_GP(app)
2
3       clearvars -except app
4       clc, format long
5
6       %setting error parameter as feedback
7       error=1;
8       summary=0;
9       reports_dir=0;
10      graphs_dir=0;
11      % GETTING THE CAR DIRECTORY
12      app.statusLabel.Text = 'Loading folder';
13      app.statusLamp.Color = [0 1 1];
14      drawnow;
15      dirn = uigetdir(pwd, 'Select a folder'); %selection of directory
16      if dirn==0
17          error=104;
18          figure(app.GoTAAppUIFigure)
19      else
```

Figure 31 Main function lines 1-19

In the first part of the function there is the initialization of some variable which is not required in MATLAB, but considering the number of different variables and variables’ names used, is good caution to set the important variables to an initial value to our like.

in Moreover, in line 15 there is the string that asks to select our working folder, followed by the “IF” check to make sure that a folder has, in fact, been selected.

```
43          %DELETE OLD REPORT AND GRAPHS FOLDERS IF EXIST AND MAKE NEW ONES
44
45          template_dir=strcat(working_dir, "\templates");
46          summ_templ=strcat(template_dir, "\GP Summary.xlsx");
47          reports_dir=strcat(dirn, "\Reports");
48          graphs_dir=strcat(dirn, "\Plots");
49          skip=0;
50          tr_index=0;
51          fold1=0;
52          fold2=0;
53          fold3=0;
54
55          %Assign mark percentage values inserted from the user
56          mk_s=app.symm_mark_box.Value/100;
57          mk_d=app.dam_mark_box.Value/100;
58          mk_e=app.ex_mark_box.Value/100;
59          box_s=app.symm_cb.Value;
60          box_d=app.dam_cb.Value;
61          box_e=app.ex_cb.Value;
```

Figure 32 Main function lines 43-61

When the folder has been selected, more initializations of variable are performed, in lines 45 through 53 in particular, the paths of the .xls templates folder and the paths of graphs and reports are saved into the corresponding strings. Furthermore, five check values are initialized too.

From line 55 to 61 the are saved into the corresponding variable the values of the weighted mark selected in the application and if the box of the corresponding analysis is ticked or not.

The following lines, use the information received to set which of the three analysis has to be performed and to delete any old folder of results that may be present in the same path as the new ones.

```
136 -     for hh=1:n_tr_txt
137 -         n_tries=0;
138 -         for hhh=1:n_s3t
139 -             if contains(f(hhh), track_txt(hh))
140 -                 n_tries=n_tries+1;    %counts the tries of track 'track_txt(hh)'
141 -                 if n_tries==1
142 -                     pos=hhh;
143 -                 end
144 -             end
145 -         end
146           %if the tracks hh has enough tries, saves it in 'test_tr' and
147           %counts it
148 -         if n_tries>=1
149 -             test_tr(htr)=track_txt(hh);
150 -             pos_test_tr(htr)=pos;
151 -             n_tries_tr(htr)=n_tries;
152 -             htr=htr+1;
153 -         end
154 -     end
155       %cuts the arrays removing empty spots
156 -     test_tr=test_tr(1:htr-1);
157 -     pos_test_tr=pos_test_tr(1:htr-1);
158 -     n_tries_tr=n_tries_tr(1:htr-1);
```

Figure 33 Main functions lines 136-158

In the lines from 136 to 158, the main function compares with a double for loop all the tracks that can analyze according to its database of tracks and all the tracks present in the vehicle's folder that has been selected, moreover it counts the number of tests that are present for each track and memorizes the relative positions.

```
166 - for h=1:sfold
167 -     bfold=char(convertStringsToChars(names_dir(h)));
168 -     if and(contains(bfold,"damage"),box_d==1)
169 -         d = paths_dir(h);
170 -         [track_names, mark, error]=b_average_damage(d, template_dir, reports_dir, test_tr, error, app, n_tries_tr);
171 -         [n_tr, ~]=size(track_names);    %damage subfunction also gives a confirmation of the number of tracks to be analysed
172 -         summ=zeros(n_tr,12);    %creates the summary mark matrix
173 -         summ(:,2:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the damage marks into summ matrix
174 -         mark=0;
175
176 -     elseif and(contains(bfold,"levelcrossing"),box_s==1)
177 -         d = paths_dir(h);
178 -         [mark, error]=a_symmetry(d, template_dir, reports_dir, test_tr, graphs_dir, error, app, n_tries_tr);
179 -         summ(:,1:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the symmetry marks into summ matrix
180 -         mark=0;
181
182 -     elseif and(contains(bfold,"execution"),box_e==1)
183 -         d = paths_dir(h);
184 -         [mark, error]=c_execution(d, template_dir, reports_dir, test_tr, graphs_dir, error, app, n_tries_tr);
185 -         summ(:,3:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the execution marks into summ matrix
186 -         mark=0;
187
188 -     end
189 - end
190
```

Figure 34 Main function lines 166-189

From line 166 to 189 a for loop runs through the folders in the vehicle's selected folder and checks whenever one of the is called "levelcrossing", "damage" or "execution" calls the corresponding function and proceeds with the analysis. While each of these three functions require quite a lot of variables, it returns only two or three variables. The two variables are a number called error which if it where to be different from zero would mean that a known error has occurred, and the second is the variable "mark" which is a matrix containing the marks for each track and each try of the single analysis.

Immediately after one of the three functions has ended the marks are copied into the summary matrix which collects all the data present in the final table in the application.

The only function returning three variables is the damage function which must be always performed as it guarantees a double check of the tracks to be analyzed.

```
193 -       GP=strings(n_tr,1);
194 -       for i=1:n_tr
195 -               for ii=0:n_tries_tr(i)-1
196 -                   if (mk_s>0&&mk_d>0&&mk_e>0)
197 -                       check_mark=[summ(i,ii*4+1), summ(i,ii*4+2), summ(i,ii*4+3)];
198 -                   elseif (mk_s>0&&mk_d>0)
199 -                       check_mark=[summ(i,ii*4+1), summ(i,ii*4+2)];
200 -                   elseif (mk_d>0&&mk_e>0)
201 -                       check_mark=[summ(i,ii*4+2), summ(i,ii*4+3)];
202 -                   elseif mk_d>0
203 -                       check_mark=summ(i,ii*4+2);
204 -                   end
205 -                   if and(check_mark>0, check_mark<=10)
206 -                       summ(i,ii*4+4)=(mk_s*summ(i,ii*4+1)+mk_d*summ(i,ii*4+2)+mk_e*summ(i,ii*4+3));
207 -                   end
208 -               end
209           %checks the higher mark between the weighted averages and
210           %assigns the GP try
211 -         [track_mark, high_mark]=max(summ(i,4:4:12));
212 -         if and(track_mark>0, track_mark<=10)
213 -             if high_mark==1
214 -                 GP(i)=strcat("Try 1 ", f(pos_test_tr(i)));
215 -             elseif high_mark==2
216 -                 GP(i)=strcat("Try 2 ", f(pos_test_tr(i)+1));
217 -             elseif high_mark==3
218 -                 GP(i)=strcat("Try 3 ", f(pos_test_tr(i)+2));
219 -             end
220 -         else %if no marks is between 0 and 10 there is no acceptable try
221 -             GP(i)="No acceptable try";
222 -         end
223 -     end
```

Figure 35 Main function lines 193-223

The above set of lines there is a for loop that runs through all the tracks analyzed and performs the following operations:

- Checks if all the marks for each try are in the range 0-10 and there has not been any major error;
- If the marks are in the range, it performs the weighted average accordingly to the percentages specified by the user;
- Finally ranks from 1 to 3 the highest to lowest mark try selecting the golden Pass for each track.

```
242 -                copyfile(summ_templ, reports_dir)
243 -                summ_file=strcat(reports_dir, "\GP Summary.xlsx");
244 -                if (box_s&&box_d&&box_e)==1
245 -                    summary=table(track_names, summ(:,1), summ(:,2), summ(:,3), summ(:,4), summ(:,5),
246 -                    summary.Properties.VariableNames = {'Track names', 'S1', 'D1', 'E1', 'Fin1', 'S2'
247 -                elseif (box_s&&box_d)==1
248 -                    summary=table(track_names, summ(:,1), summ(:,2), summ(:,4), summ(:,5), summ(:,6),
249 -                    summary.Properties.VariableNames = {'Track names', 'S1', 'D1', 'Fin1', 'S2', 'D2'
250 -                elseif (box_d&&box_e)==1
251 -                    summary=table(track_names, summ(:,2), summ(:,3), summ(:,4), summ(:,6), summ(:,7),
252 -                    summary.Properties.VariableNames = {'Track names', 'D1', 'E1', 'Fin1', 'D2', 'E2'
253 -                elseif box_d==1
254 -                    summary=table(track_names, summ(:,2), summ(:,4), summ(:,6), summ(:,8), summ(:,10)
255 -                    summary.Properties.VariableNames = {'Track names', 'D1', 'Fin1', 'D2', 'Fin2', 'D
256 -                end
257 -                writetable(summary,summ_file,'sheet','GP');
258 -                error=0;
259
260 -            elseif fold1==0
261 -                error=101; %Missing level crossing folder
262 -            elseif fold2==0
263 -                error=102; %Missing damage folder
264 -            elseif fold3==0
265 -                error=103; %Missing execution folder
266 -            end
267 -        end
268 -    end
```

Figure 36 Main function lines 242-268 final part

In the last part of the main function, the summary table to be shown in the application is built and is exported a copy into the report file too.

At the very bottom there are the lines dedicated to possible errors that could arise.

7.1.1. Main report

Track names	S1	D1	E1	Fin1	S2	D2	E2	Fin2	S3	D3	E3	Fin3	Golden Pass try
AV	5.83	10	7.42	8.13	5.83	4.17	7.31	4.92	8.33	5.83	7.21	6.96	Try 1 M182_BEV__VP124_FPL_BPG_AV_140.csv
MCISA	5.33	5	5.31	5.15	8.88	6.67	7.06	7.66	5.79	8.33	6.35	7.19	Try 2 M182_BEV__VP124_FPL_BPG_MCISA_134.csv
OB-PROC	6.54	6.67	7.41	6.61	7.79	4.17	6.96	5.8	5.67	9.17	6.81	7.59	Try 3 M182_BEV__VP124_FPL_BPG_OB-PROC_159.csv
OL-DS	7.58	5.83	7.92	6.62	3.83	8.33	7.9	6.31	8.58	5.83	8.18	7.07	Try 3 M182_BEV__VP124_FPL_BPG_OL-DS_112.csv
PABA	7.21	9.17	8.3	8.29	6.25	6.67	7.61	6.48	6.54	4.17	7.54	5.24	Try 1 M182_BEV__VP124_FPL_BPG_PABA_096.csv
PLX	7.25	5	8.12	6.01	7.25	9.17	7.63	8.3	5.5	5.83	7.57	5.68	Try 2 M182_BEV__VP124_FPL_BPG_PLX_123.csv
PORF-PROC	8.08	5.83	8.35	6.85	3.83	5.83	7.75	4.93	8.08	8.33	7.97	8.22	Try 3 M182_BEV__VP124_FPL_BPG_PORF-PROC_115.csv
POT	5.25	8.33	7.98	6.95	8.88	4.17	7.36	6.29	5.88	7.5	8.83	6.77	Try 1 M182_BEV__VP124_FPL_BPG_POT_163.csv
RAL	4.75	5.83	6.58	5.35	8.46	7.5	7.12	7.93	6.79	6.67	6.21	6.72	Try 2 M182_BEV__VP124_FPL_BPG_RAL_158.csv
TR-BU	6.67	10	7.8	8.5	6.42	5	8	5.64	6.92	5	8.31	5.86	Try 1 M182_BEV__VP124_FPL_BPG_TR-BU_109.csv

The main report called “GP_summary” is exactly the table shown in the application but is saved with the other reports for ease of access.

7.2. Symmetry

Since the following functions are full of ordering lines which are not crucial for the understanding of the analysis, just the most important parts will be shown. For any further information the complete codes are attached.

The symmetry function starts with a thorough ordering of all the channels’ names for each try of each track in couples of symmetric channels. Once the couples are formed it is easier to analyze them. Of course, through the name the function also save the position and the path of every single file.

```
351 -   while i<m   %i up to m that is the number of coupled channels
352
353         %import of datas
354
355 -       ChanL=readtable(f_ord{i},'NumHeaderLines',14);
356 -       ChanL=ChanL{:,:};
357 -       accL=ChanL(:,1);    %acc is the array with accelerometer values
358 -       nL=ChanL(:,2);      %n is the array with number of level crossing repetitions
359
360 -       ChanR=readtable(f_ord{i+1},'NumHeaderLines',14);
361 -       ChanR=ChanR{:,:};
362 -       accR=ChanR(:,1);
363 -       nR=ChanR(:,2);
364         %for values measured on the Y axis the accelerometers
365         %values for the right side change sign
366 -       ctY=or(contains(Ch_n_ord(i),"FY"),contains(Ch_n_ord(i),"AccY"));
367         %for simplicity the values are copied into x and y
368         %arrays named for the cartesian coordinates they will
369         %be shown in the graphs
370 -       if(ctY==1)
371 -           xR=-1*accR;
372 -       else
373 -           xR=accR;
374 -       end
375 -       xL=accL;
376 -       yL=nL;
377 -       yR=nR;
```

Figure 38 Symmetry function lines 351-377

From line 351 starts the actual symmetry analysis of the channels. For each of the three tries of each track, the function runs through the couples and for each couple saves the needed values to perform the symmetry analysis, with a particular attention if the couple analyzed corresponds to channels on the Y-axis of the vehicle in which case one of the two data is reversed otherwise there would be two specular curves instead of coincident.

```
379         %SPLINE EVALUATION, SUBTRACTION OF THE TWO CHANNELS AND
380         %SELECTION OF MAXIMA AND MINIMA
381
382 -       pR=spline(xR,yR);
383 -       pL=spline(xL,yL);
384 -       x=linspace(max(min(xR),min(xL)),min(max(xR),max(xL)),div);
385 -       yL1=ppval(pL,x);
386 -       yR1=ppval(pR,x);
387 -       ydiff=yL1-yR1;
388
389 -       mxR(j,c)=max(accR);
390 -       mxL(j,c)=max(accL);
391 -       mnR(j,c)=min(accR);
392 -       mnL(j,c)=min(accL);
```

Figure 37 Symmetry function lines 379-392

In the above set of lines, the piecewise polynomial approximation of the level crossing points is performed through the spline function. A new set of values equals for both left and right channel is created and using the ppval function the corresponding y-axis values are saved in the arrays “yL1” and “yR1”.

Subsequently the difference among the two new arrays is performed and the maxima and minima of each channel are stored into “mxR, mxL, mnR, mnL”

```
394          %GRAPH CREATION AND SAVING
395
396 -        title_symm=strcat(f_names(i),"_", Channel_name(j), " symm");
397 -        fig = figure('visible','off');
398 -        plot(x,yLl,'--','color','r');
399 -        hold on;
400 -        plot(x,yRl,'--','color','b');
401 -        hold on;
402 -        plot(x,ydiff,'-','color','g','LineWidth',1);
403 -        title(title_symm,'Interpreter','none');
404 -        ylabel('level crossing divisions');
405 -        xlabel('acceleration [m/s^2]');
406 -        legend('Left','Right', 'Difference','location','best');
407 -        hold off;
408 -        figname=strcat(title_symm, '.fig');
409 -        savefig(fig, fullfile(plotsdir_symm, figname));
410 -        close (fig)
411          %updates the loading percentage
412 -        graph_count=graph_count+1;
413 -        perc_symm=round(graph_count*100/(n_couples*n_tracks*max(n_tries_tr)),0);
414 -        percentage_status=char(strcat("Symmetry ", string(perc_symm), "%"));
415 -        app.statusLabel.HorizontalAlignment = 'left';
416 -        app.statusLabel.Text = percentage_status;
```

Figure 39 Symmetry function lines 394-416

Lines 394 to 416 are necessary to save the graphs in the graphs' folder and also adjust the progress percentage shown in the application.

An example of a symmetry analysis graph is presented below.

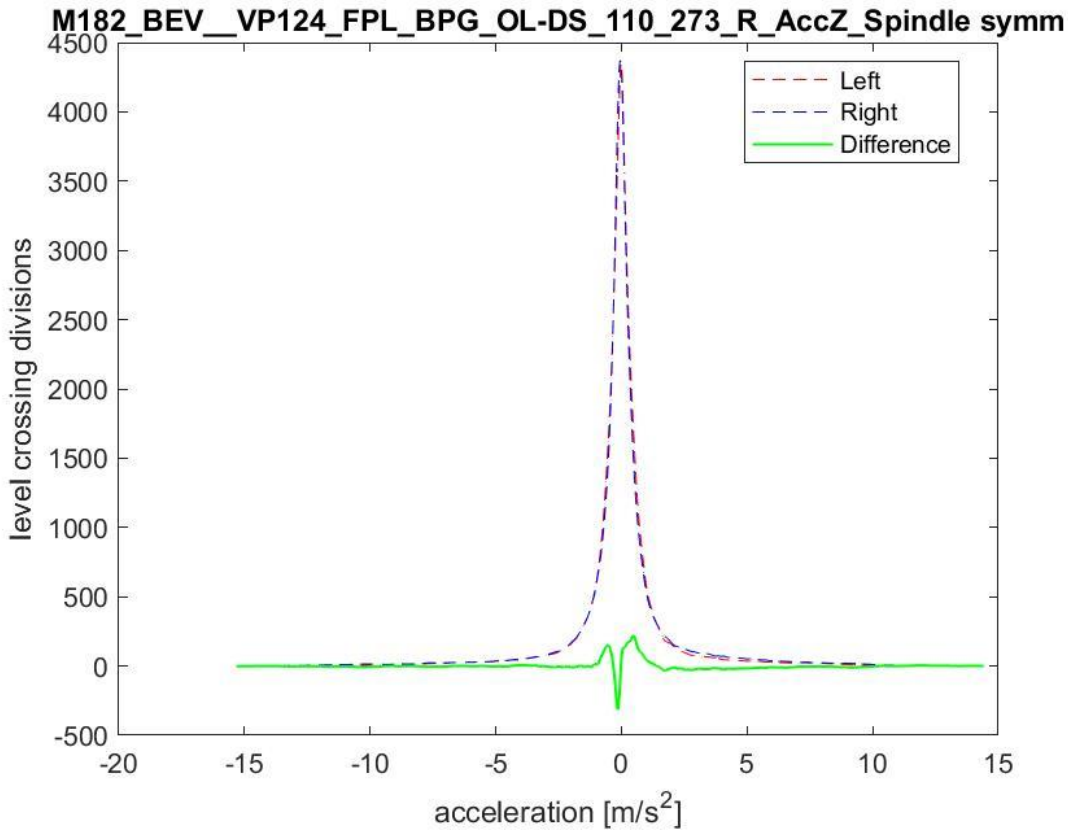


Figure 40 Symmetry graph

```
418          %EVALUATION OF STANDARD DEVIATION AND INTEGRALS
419 -        int_diff=0;
420 -        int=0;
421          %for loop evaluates the absolute value of the integral
422          %of the difference spline
423 -        for e=1:div-1
424 -            int_diff=int_diff+abs(trapz(ydiff(e:e+1)));
425 -            int=int+abs(trapz(yL1(e:e+1)));
426 -        end
427
428          %save the values of integral of difference spline and
429          %the differences between maxima and minima into a three
430          %coloumns matrix in order to compare the results among
431          %the three tries
432 -        stdev(j,c)=std(ydiff);
433 -        tint_diff(j,c)=int_diff;
434 -        tint(j,c)=int;
435 -        mx_diff(j,c)=abs(mxR(j,c)-mxL(j,c))/max(mxR(j,c),mxL(j,c));
436 -        mn_diff(j,c)=abs(mnR(j,c)-mnL(j,c))/max(abs(mnR(j,c)),abs(mnL(j,c)));
437 -        int_perc(j,c)=tint_diff(j,c)/tint(j,c);
438 -        j=j+1;
439 -        i=i+2;
440 -    end
```

Figure 41 Symmetry function lines 418-440

Precisely after the graph creation, there is the set of lines dedicated to the actual parameters that decide the marks of each test.

In line 423 there is the for loop that calculates the cumulated integral of the y_difference curve.

The first parameters to be calculated is “stdev” which is an array storing all the standard deviations of the different tries and different tracks.

Following there are “mx_diff” and “mn_diff” which are the differences in maxima and minima of the two left and right channels. The difference is calculated as a percentage of the maximum or the minimum value in order to be comparable with the results of the other channels.

Finally, there is the last parameter that is the cumulated integral of the difference signal. This too is normalized by dividing it by the integral of the left signal.

7.2.1. Symmetry report

Couple n.	Couple name	St dev 1	Ptsdev1	Max diff 1	Ptmx1	Min diff 1	Ptmn1	Int % 1	Ptint1
1	F_AccZ_Spindle	70,630	2	0,243	1	0,142	3	0,118	2
2	R_AccZ_Spindle	38,196	2	0,037	3	0,055	3	0,080	3

St dev 2	Ptsdev2	Max diff 2	Ptmx2	Min diff 2	Ptmn2	Int % 2	Ptint2	St dev 3	Ptsdev3	Max diff 3	Ptmx3	Min diff 3	Ptmn3	Int % 3	Ptint3
83,669	1	0,214	2	0,197	2	0,148	1	59,811	3	0,172	3	0,222	1	0,112	3
61,308	1	0,055	2	0,177	2	0,141	1	30,128	3	0,094	1	0,207	1	0,086	2

Figure 42 Symmetry report example

After the calculations, points from 1 to 3 are awarded from the worst to the best try in for each of the for parameters, then the weighted average according to the percentages aforementioned is obtained and the results are saved in the summary matrix and exported in the symmetry report which is shown above.

7.3. Average damage

The average damage analysis has the shortest function as it is completely performed in a matrix.

```
195 -         for i=1:n_tries_tr(m)
196 -             b=char(convertStringsToChars(f((m-1)*n_tries_tr(m)+i)));
197 -             dmg1=readtable(b);  %import file
198               %split the table in number of channel, name and damage value
199 -             Ch_n_prov=table2array(dmg1(:,1));
200 -             Ch_name_prov=table2array(dmg1(:,2));
201 -             Dam_prov=string(table2array(dmg1(:,3)));
202 -             [n_ch_prov, ~]=size(Ch_n_prov);
203 -             jj=1;
204 -             n_ch=0;
205
206               %CHECK OF THE CHANNELS TO BE EVALUATED AND COUNTING
207
208 -             for ii=1:n_ch_prov
209 -                 if contains(Ch_name_prov(ii), dam_ch)
210 -                     Ch_n(jj)=Ch_n_prov(ii);
211 -                     Ch_name(jj)=Ch_name_prov(ii);
212 -                     Dam(jj,:)=Dam_prov(ii,:);
213 -                     jj=jj+1;
214 -                     n_ch=n_ch+1;
215 -                 end
216 -             end
217 -             M_prov(:,j)=Dam;
218 -             j=j+3;
219 -         end
```

Figure 43 Damage function lines 195-219

Instead of using a read function column by column in this case a readtable function is used which collect the data all at once and is split afterwards into the different columns.

From line 208 to 216 a for cycle checks the actual number of channels corresponding to the requirements for the performance of the average damage analysis.

```
228 -         for i=1:n_ch    %for each channel
229 -             M(i,wM-1)=mean(M(i,1:3:7)); %mean of damage value
230 -             for j=0:n_tries_tr(m)-1   %for each channel and for each try
231 -                 M(i,j*3+2)=M(i,j*3+1)/M(i,wM-1);    %damage over average damage
232 -                 M(i,j*3+3)=((M(i,j*3+1)/M(i,wM-1))-1)^2;    %standard deviation
233 -                 M(i,wM)=M(i,wM)+M(i,j*3+3);     %sum of standard deviation
234 -             end
235 -         end
```

Figure 44 Damage function lines 228-235

Working with plain numbers and a matrix it is easy to perform every calculation needed using a double for cycle.

The parameters calculated are average damage and standard deviation.

After the calculation the points are assigned in a similar way with respect to the symmetry analysis and a report is compiled and exported as well.

7.3.1. Average damage report

Channel name	Chan N	Dam1	Dam1/Avg Dam	Pt dam1	$(D1/A-1)^2$	Pt sdev1		
LF_AccZ_Spindle	27	0,010635	0,971883758	3	0,000790523	3		
RF_AccZ_Spindle	30	0,011017	0,849116226	1	0,022765913	1		
LR_AccZ_Spindle	33	0,0096425	0,910887192	2	0,007941093	2		
RR_AccZ_Spindle	36	0,012722	0,826354306	1	0,030152827	1		
		Dam2	Dam2/Avg Dam	Pt dam2	$(D2/A-1)^2$	Pt sdev2		
		0,010271	0,938619471	2	0,003767569	2		
		0,014133	1,089276539	2	0,0079703	2		
		0,010495	0,99141935	3	7,36276E-05	3		
		0,016403	1,065452735	3	0,00428406	3		
		Dam3	Dam3/Avg Dam	Pt dam3	$(D3/A-1)^2$	Pt sdev3	Avg damage	$\Sigma(D/A-1)^2$
		0,011922	1,089496771	1	0,008009672	1	0,010942667	0,012567764
		0,013774	1,061607235	3	0,003795451	3	0,012974667	0,034531665
		0,01162	1,097693458	1	0,009544012	1	0,010585833	0,017558732
		0,017061	1,108192959	2	0,011705716	2	0,015395333	0,046142604

Figure 45 Damage report example

7.4. Execution

The final function is the execution function. The most difficult task to perform in the execution analysis was to pair the vehicle's coordinates with the start and finish of each track. As for the first two analysis, just the main part of the script will be discuss and in particular, since the execution has a different script for each track, it will be discussed only one track.

```
1050 -          elseif sum(and(contains(test_tr,"OLC-LAS"),contains(b,"OLC-LAS")))==1
1051
1052 -              nlt=zeros(4,1);
1053 -              nlg=zeros(4,1);
1054               %the points in the time serie with the closest longitude and
1055               %latitude values compared to fixed coordinates are taken
1056 -              [~, mn_olclas]=min(lt);
1057 -              lt_prov=lt(1:mn_olclas);
1058 -              lg_prov=lg(1:mn_olclas);
1059
1060 -              [~, nlt(1)]=min(abs(lt_prov-lt_olclas(1)));
1061 -              [~, nlt(2)]=min(abs(lt_prov-lt_olclas(2)));
1062 -              [~, nlg(1)]=min(abs(lg_prov-lg_olclas(1)));
1063 -              [~, nlg(2)]=min(abs(lg_prov-lg_olclas(2)));
1064
1065 -              lt_prov=lt(mn_olclas:end);
1066 -              lg_prov=lg(mn_olclas:end);
1067 -              [~, nlt(3)]=min(abs(lt_prov-lt_olclas(3)));
1068 -              [~, nlt(4)]=min(abs(lt_prov-lt_olclas(4)));
1069 -              [~, nlg(3)]=min(abs(lg_prov-lg_olclas(3)));
1070 -              [~, nlg(4)]=min(abs(lg_prov-lg_olclas(4)));
1071 -              nlt(3)=nlt(3)+mn_olclas;
1072 -              nlt(4)=nlt(4)+mn_olclas;
1073 -              nlg(3)=nlg(3)+mn_olclas;
1074 -              nlg(4)=nlg(4)+mn_olclas;
```

Figure 46 Execution function lines 1050-1074

The OLC-LAS track was considered. In these first lines of the function, the crucial points of the time series where the analysis starts, ends and/or changes have to be found.

This is done by comparing the latitude and longitude of each point with salient coordinates of the track; the points with the closest coordinates are selected and stored.

```
1085            %for loop runs through the points of the test
1086            %checks that the speed is in the limits
1087            %sets that the test is failed
1088            %saves the failing points
1089            %counts the failing points
1090 -          for j=nlt(1):nlt(2)
1091 -              if or(v(j)<v_down(j),v(j)>v_up(j))
1092 -                  fail_vect(i)=1;
1093 -                  fail_points(j)=t(j);
1094 -                  fail_count(i)=fail_count(i)+1;
1095 -              end
1096 -          end
1097 -          for j=nlt(3):nlt(4)
1098 -              if or(v(j)<v_down(j),v(j)>v_up(j))
1099 -                  fail_vect(i)=1;
1100 -                  fail_points(j)=t(j);
1101 -                  fail_count(i)=fail_count(i)+1;
1102 -              end
1103 -          end
```

Figure 47 Execution function lines 1085-1103

Once the salient points are selected and the upper and lower limit speeds are stored in place, a for loop runs through every single point of the acquisition and ensures that there is no point outside of the given range. If any point is out, its position is stored in the "fail_points" array.

```
1105            %creates graph and saves it
1106 -          graph_t=strcat(testname(i));
1107 -          fig = figure('visible','off');
1108 -          plot(t,v,'--','color','b')
1109 -          hold on
1110 -          plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
1111 -          hold on
1112 -          plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
1113 -          hold on
1114 -          plot(t(nlt(3):nlt(4)),v_up(nlt(3):nlt(4)),'--','color','g')
1115 -          hold on
1116 -          plot(t(nlt(3):nlt(4)),v_down(nlt(3):nlt(4)),'--','color','g')
1117 -          hold on
1118 -          for k=1:size(n)
1119 -              if fail_points(k)~=0
1120 -                  plot(fail_points(k),v(k),'x','color','r')
1121 -              end
1122 -          end
1123 -          title(graph_t);
1124 -          ylabel('Speed [km/h]');
1125 -          xlabel('Time [s]');
1126 -          legend('Speed','Upper limit', 'Lower limit','location','best')
1127 -          hold off
1128 -          figname=strcat(graph_t, '.fig');
1129 -          savefig(fig, fullfile(plotsdir_ex, figname));
1130 -          close (fig)
```

Figure 48 Execution function lines 1105-1130

These lines produce and save the graph of the speed. As in the symmetry analysis they are saved in the graphs' folder. The graphs show not only the actual speed, but also upper and lower limits and also highlight the fail points if any are present.

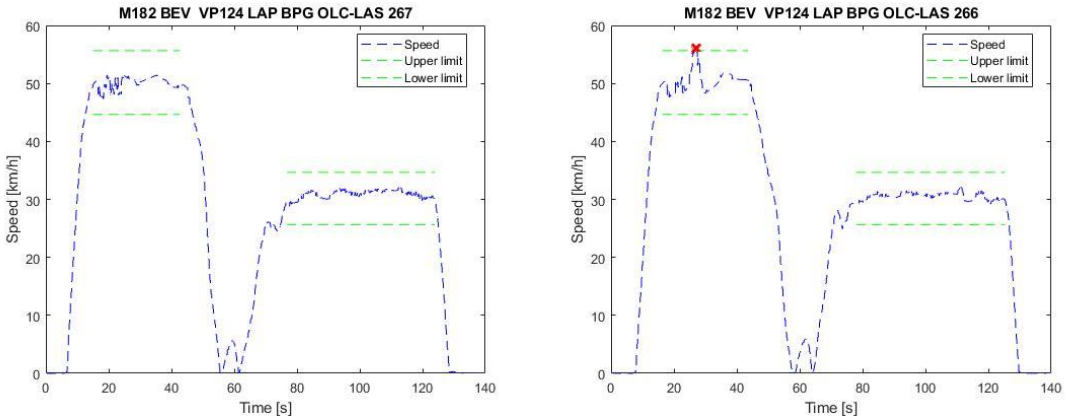


Figure 50 Execution graphs example

```
1133 -          if or(fail_vect(i)==0,fail_vect(i)==1)
1134 -              for j=nlt(1):nlt(2)
1135 -                  if v_ideal(j)==0
1136 -                      std_low(j)=0;
1137 -                      std_dev(j)=0;
1138 -                  else
1139 -                      std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
1140 -                      std_dev(j)=(v(j)/v_ideal(j)-1)^2;
1141 -                  end
1142 -              end
1143 -              for j=nlt(3):nlt(4)
1144 -                  if v_ideal(j)==0
1145 -                      std_low(j)=0;
1146 -                      std_dev(j)=0;
1147 -                  else
1148 -                      std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
1149 -                      std_dev(j)=(v(j)/v_ideal(j)-1)^2;
1150 -                  end
1151 -              end
1152                  %calculates the mean among all the tested points of the
1153                  %standard deviation
1154 -              ev_points(i)=nlt(4)-nlt(3)+nlt(2)-nlt(1)+2;
1155 -              testpoints=zeros(ev_points(i),1);
1156 -              testpoints(1:(nlt(2)-nlt(1))+1)=std_dev(nlt(1):nlt(2));
1157 -              testpoints((nlt(2)-nlt(1))+2:end)=std_dev(nlt(3):nlt(4));
1158 -              testpoints_low=zeros(nlt(4)-nlt(3)+nlt(2)-nlt(1)+2,1);
1159 -              testpoints_low(1:(nlt(2)-nlt(1))+1)=std_low(nlt(1):nlt(2));
1160 -              testpoints_low((nlt(2)-nlt(1))+2:end)=std_low(nlt(3):nlt(4));
1161 -              std_dev_tr(i)=mean(testpoints)^0.5;
1162 -              std_dev_low(i)=mean(testpoints_low)^0.5;
1163 -          end
```

Figure 49 Execution function lines 1133-1163

In the last part of the single track execution function, the standard deviation from the ideal speed is calculated and once all the tracks are done, the marks are calculated remembering that a penalty of up to 40% of the mark may be applied accordingly to the following rules:

- Less than 2% of points out of range: full mark;
- Between 2% and 5% of points out of range: 20% reduction;
- Between 5% and 10% of points out of range: 40% reduction.

7.4.1. Execution report

As for the other analysis the execution function has its own report too, but it has a different scheme with respect to the other two.

Test name	Result	Standard deviation	Mark
M182 BEV VP124 FPL BPG MCISA 133	Pass	7,45%	5,31
M182 BEV VP124 FPL BPG MCISA 134	Pass	4,66%	7,06
M182 BEV VP124 FPL BPG MCISA 160	Pass	5,79%	6,35
M182 BEV VP124 FPL BPG OB-PROC 132	Pass	8,91%	7,41
M182 BEV VP124 FPL BPG OB-PROC 155	Pass	10,50%	6,96
M182 BEV VP124 FPL BPG OB-PROC 159	Pass	10,95%	6,81
M182 BEV VP124 FPL BPG OL-DS 110	Pass	2,35%	7,92
M182 BEV VP124 FPL BPG OL-DS 111	Pass	2,37%	7,90
M182 BEV VP124 FPL BPG OL-DS 112	Pass	2,06%	8,18
M182 BEV VP124 FPL BPG PABA 096	Pass	2,68%	8,30
M182 BEV VP124 FPL BPG PABA 103	Pass	3,76%	7,61
M182 BEV VP124 FPL BPG PABA 113	Pass	3,88%	7,54

Figure 51 Execution report example

8. CONCLUSIONS

Considering the radical change of the method by which the Golden Pass choice is made, is reasonable that the creation of an application from scratch would undergo a lot of adjustment steps.

Before reaching the last version of the application a total of eleven sets of functions were created because of the continuous improvement in precision required and subsequently due to the error which arose from the application testing.

The results are really promising mainly because of the time gained to complete the Golden Pass selection but also because of the coherence that the results have with the visual analysis.

The analysis with the most difficulties is definitely the execution analysis as the tests in each track in some cases require a double passage on the same GPS coordinates and there are not only straight paths, this multiple passages on the same coordinates often create confusion and problems arose easily during the testing.

9. BIBLIOGRAPHY

[1] International Journal of Research Publication and Reviews, Vol 3, no 12, pp 1920-1923, December 2022 1922

[2] S. Holm and J. de Mare', A Simple Model for Fatigue Life, IEEE Transactions on Reliability, 1988

[3] S. Holm,[a] J. de Mare,[a] T. Svensson[b], Prediction of fatigue life based on level crossing and load history, (a) Mathematical statistics, Chalmer University of technology, S-412 96 Göteborg, Sweden, (b) Material and mechanics, Swedish National Testing and Research Institute, Box 857, S-501 15 Boras, Sweden

[4] A. S. and V. S., "STRUCTURAL DURABILITY ROAD TEST IN BALOCCO PROVING GROUND", FCA ITALY - V. C. & I. - VEHICLE VALIDATION & DURABILITY, 2017

[5] M. A. and K. L., "STRUCTURAL DURABILITY ROAD TEST DAQ ON TEST TRACK", FCA ITALY - V. C. & I. - VEHICLE VALIDATION & DURABILITY, 2019

10. ATTACHMENTS

10.1. GoTA application code

```
classdef GP_App_exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        GoTAAppUIFigure              matlab.ui.Figure
        GridLayout                   matlab.ui.container.GridLayout
        LeftPanel                    matlab.ui.container.Panel
        StartAnalysisButton          matlab.ui.control.Button
        statusLamp                   matlab.ui.control.Lamp
        OpenfigureButton             matlab.ui.control.Button
        statusLabel                  matlab.ui.control.Label
        symm_cb                      matlab.ui.control.CheckBox
        dam_cb                       matlab.ui.control.CheckBox
        ex_cb                        matlab.ui.control.CheckBox
        MarkLabel                    matlab.ui.control.Label
        ChannelselectionSwitchLabel  matlab.ui.control.Label
        ChannelselectionSwitch       matlab.ui.control.Switch
        OpenxlsreportsButton         matlab.ui.control.Button
        symm_mark_box                matlab.ui.control.NumericEditField
        dam_mark_box                 matlab.ui.control.NumericEditField
        ex_mark_box                  matlab.ui.control.NumericEditField
        Reset                        matlab.ui.control.Button
        CenterPanel                  matlab.ui.container.Panel
        UITable                      matlab.ui.control.Table
        TextArea                     matlab.ui.control.TextArea
        ListBox                      matlab.ui.control.ListBox
        RightPanel                   matlab.ui.container.Panel
    end

    % Properties that correspond to apps with auto-reflow
    properties (Access = private)
        onePanelWidth = 576;
        twoPanelWidth = 768;
    end


    properties (Access = private)
        reports_dir % Description
        graphs_dir % Description
        valid % Description
    end

    methods (Access = private)

    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: StartAnalysisButton
        function StartAnalysisButtonPushed(app, event)
            if
```

```
(app.ex_mark_box.Value+app.dam_mark_box.Value+app.symm_mark_box.Value)==100
                app.TextArea.Visible = 'off';
                app.UITable.Visible = 'off';
                app.symm_mark_box.Enable = 'off';
                app.dam_mark_box.Enable = 'off';
                app.ex_mark_box.Enable = 'off';
                app.symm_cb.Enable = 'off';
                app.dam_cb.Enable = 'off';
                app.ex_cb.Enable = 'off';
                app.StartAnalysisButton.Enable = 'off';
                app.OpenfigureButton.Enable = 'off';
                app.OpenxlsreportsButton.Enable = 'off';
                drawnow;
                [summary, error, reports_dir, graphs_dir]=main_GP(app); %#ok<*ADPROPLC>
                app.reports_dir=reports_dir;
                app.graphs_dir=graphs_dir;
                app.symm_mark_box.Enable = 'on';
                app.dam_mark_box.Enable = 'on';
                app.ex_mark_box.Enable = 'on';
                app.symm_cb.Enable = 'on';
                app.dam_cb.Enable = 'on';
                app.ex_cb.Enable = 'on';
                app.StartAnalysisButton.Enable = 'on';
                if app.valid==1
                    app.OpenfigureButton.Enable = 'on';
                    app.OpenxlsreportsButton.Enable = 'on';
                end
                if error==0
                    app.valid=1;
                    app.UITable.Visible = 'on';
                    app.UITable.ColumnName=summary.Properties.VariableNames;
                    app.UITable.Data=summary;
                    app.statusLamp.Color = [0 1 0];
                    app.statusLabel.Text = 'Done';
                    app.OpenfigureButton.Enable = 'on';
                    app.OpenxlsreportsButton.Enable = 'on';
                    drawnow;
                elseif error==101
                    app.statusLamp.Color = [1 0 0];
                    app.statusLabel.Text = 'Error';
                    app.TextArea.Visible = 'on';
                    app.TextArea.Value = 'Missing level crossing folder';
                elseif error==102
                    app.statusLamp.Color = [1 0 0];
                    app.statusLabel.Text = 'Error';
                    app.TextArea.Visible = 'on';
                    app.TextArea.Value = 'Missing damage folder';
                elseif error==103
                    app.statusLamp.Color = [1 0 0];
                    app.statusLabel.Text = 'Error';
                    app.TextArea.Visible = 'on';
                    app.TextArea.Value = 'Missing execution folder';
                elseif error==104
                    app.statusLamp.Color = [1 0 0];
                    app.statusLabel.Text = 'Error';
                    app.TextArea.Visible = 'on';
                    app.TextArea.Value = 'No folder selected';
                end
            else
```

```
                app.statusLamp.Color = [1 0 0];
                app.statusLabel.Text = 'Error';
                app.TextArea.Visible = 'on';
                app.TextArea.Value = 'The sum of the percentages must be 100';
            end
        end

        % Display data changed function: UITable
        function UITableDisplayDataChanged(app, event)
            newDisplayData = app.UITable.DisplayData;

        end

        % Button pushed function: OpenfigureButton
        function OpenfigureButtonPushed(app, event)
            [fig_open, fig_open_path]=uigetfile('*.fig',"Select one or more figure to
display",strcat(app.graphs_dir,"\"),"MultiSelect","on");
            if iscell(fig_open)
                fig_open=strcat(fig_open_path,"\",fig_open);
                fig_open=string(fig_open);
                [~, sz_open]=size(fig_open);
                for i_open=1:sz_open
                    openfig(fig_open(1,i_open),"visible");
                end
            elseif fig_open~=0
                fig_open=strcat(fig_open_path,"\",fig_open);
                fig_open=string(fig_open);
                [~, sz_open]=size(fig_open);
                for i_open=1:sz_open
                    openfig(fig_open(1,i_open),"visible");
                end
            end
        end

        % Value changed function: ex_cb
        function ex_cbValueChanged(app, event)
            value = app.ex_cb.Value;
            if value==0
                app.ex_mark_box.Value=0;
                app.ex_mark_box.Editable = 'off';
            elseif value==1
                app.ex_mark_box.Editable = 'on';
            end
        end

        % Value changed function: dam_cb
        function dam_cbValueChanged(app, event)
            value = app.dam_cb.Value;
            if value==0
                app.dam_cb.Value=1;
                app.TextArea.Visible = 'on';
                app.TextArea.Value = 'The damage analysis must be performed';
            end
        end

        % Value changed function: symm_cb
        function symm_cbValueChanged(app, event)
            value = app.symm_cb.Value;
            if value==0
```

```
                app.symm_mark_box.Value=0;
                app.symm_mark_box.Editable = 'off';
            elseif value==1
                app.symm_mark_box.Editable = 'on';
            end
        end

        % Button pushed function: OpenxlsreportsButton
        function OpenxlsreportsButtonPushed(app, event)
            [rep_open, rep_open_path]=uigetfile('*.xlsx',"Select one or more report to
open",strcat(app.reports_dir,"\"),"MultiSelect","on");
            if iscell(rep_open)
                rep_open=strcat(rep_open_path,"\",rep_open);
                rep_open=string(rep_open);
                [~, sz_open]=size(rep_open);
                for i_open=1:sz_open
                    winopen(rep_open(1,i_open));
                end
            elseif rep_open~=0
                rep_open=strcat(rep_open_path,"\",rep_open);
                rep_open=string(rep_open);
                [~, sz_open]=size(rep_open);
                for i_open=1:sz_open
                    winopen(rep_open(1,i_open));
                end
            end
        end

        % Button pushed function: Reset
        function ResetButtonPushed(app, event)
            app.symm_mark_box.Value = 45;
            app.dam_mark_box.Value = 55;
            app.ex_mark_box.Value = 0;
            app.symm_cb.Value = 1;
            app.dam_cb.Value = 1;
            app.ex_cb.Value = 0;
        end

        % Changes arrangement of the app based on UIFigure width
        function updateAppLayout(app, event)
            currentFigureWidth = app.GoTAAppUIFigure.Position(3);
            if(currentFigureWidth <= app.onePanelWidth)
                % Change to a 3x1 grid
                app.GridLayout.RowHeight = {363, 363, 363};
                app.GridLayout.ColumnWidth = {'1x'};
                app.CenterPanel.Layout.Row = 1;
                app.CenterPanel.Layout.Column = 1;
                app.LeftPanel.Layout.Row = 2;
                app.LeftPanel.Layout.Column = 1;
                app.RightPanel.Layout.Row = 3;
                app.RightPanel.Layout.Column = 1;
            elseif (currentFigureWidth > app.onePanelWidth && currentFigureWidth <=
app.twoPanelWidth)
                % Change to a 2x2 grid
                app.GridLayout.RowHeight = {363, 363};
                app.GridLayout.ColumnWidth = {'1x', '1x'};
                app.CenterPanel.Layout.Row = 1;
                app.CenterPanel.Layout.Column = [1,2];
                app.LeftPanel.Layout.Row = 2;
```

```
                app.LeftPanel.Layout.Column = 1;
                app.RightPanel.Layout.Row = 2;
                app.RightPanel.Layout.Column = 2;
            else
                % Change to a 1x3 grid
                app.GridLayout.RowHeight = {'1x'};
                app.GridLayout.ColumnWidth = {145, '1x', 12};
                app.LeftPanel.Layout.Row = 1;
                app.LeftPanel.Layout.Column = 1;
                app.CenterPanel.Layout.Row = 1;
                app.CenterPanel.Layout.Column = 2;
                app.RightPanel.Layout.Row = 1;
                app.RightPanel.Layout.Column = 3;
            end
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create GoTAAppUIFigure and hide until all components are created
            app.GoTAAppUIFigure = uifigure('Visible', 'off');
            app.GoTAAppUIFigure.AutoResizeChildren = 'off';
            app.GoTAAppUIFigure.Position = [100 100 849 363];
            app.GoTAAppUIFigure.Name = 'MATLAB App';
            app.GoTAAppUIFigure.SizeChangedFcn = createCallbackFcn(app,
@updateAppLayout, true);

            % Create GridLayout
            app.GridLayout = uigridlayout(app.GoTAAppUIFigure);
            app.GridLayout.ColumnWidth = {145, '1x', 12};
            app.GridLayout.RowHeight = {'1x'};
            app.GridLayout.ColumnSpacing = 0;
            app.GridLayout.RowSpacing = 0;
            app.GridLayout.Padding = [0 0 0 0];
            app.GridLayout.Scrollable = 'on';

            % Create LeftPanel
            app.LeftPanel = uipanel(app.GridLayout);
            app.LeftPanel.Layout.Row = 1;
            app.LeftPanel.Layout.Column = 1;

            % Create StartAnalysisButton
            app.StartAnalysisButton = uibutton(app.LeftPanel, 'push');
            app.StartAnalysisButton.ButtonPushedFcn = createCallbackFcn(app,
@StartAnalysisButtonPushed, true);
            app.StartAnalysisButton.Position = [13 333 120 22];
            app.StartAnalysisButton.Text = 'Start Analysis';

            % Create statusLamp
            app.statusLamp = uilamp(app.LeftPanel);
            app.statusLamp.Tooltip = {''};
            app.statusLamp.Position = [120 307 20 20];
            app.statusLamp.Color = [0 1 1];

            % Create OpenfigureButton
```

```
            app.OpenfigureButton = uibutton(app.LeftPanel, 'push');
            app.OpenfigureButton.ButtonPushedFcn = createCallbackFcn(app,
@OpenfigureButtonPushed, true);
            app.OpenfigureButton.Enable = 'off';
            app.OpenfigureButton.Position = [13 51 120 22];
            app.OpenfigureButton.Text = 'Open figure';

            % Create statusLabel
            app.statusLabel = uilabel(app.LeftPanel);
            app.statusLabel.HorizontalAlignment = 'center';
            app.statusLabel.Position = [6 306 115 22];
            app.statusLabel.Text = 'Ready';

            % Create symm_cb
            app.symm_cb = uicheckbox(app.LeftPanel);
            app.symm_cb.ValueChangedFcn = createCallbackFcn(app, @symm_cbValueChanged,
true);
            app.symm_cb.Text = 'Symmetry';
            app.symm_cb.Position = [6 259 76 22];
            app.symm_cb.Value = true;

            % Create dam_cb
            app.dam_cb = uicheckbox(app.LeftPanel);
            app.dam_cb.ValueChangedFcn = createCallbackFcn(app, @dam_cbValueChanged,
true);
            app.dam_cb.Text = 'Damage';
            app.dam_cb.Position = [6 238 67 22];
            app.dam_cb.Value = true;

            % Create ex_cb
            app.ex_cb = uicheckbox(app.LeftPanel);
            app.ex_cb.ValueChangedFcn = createCallbackFcn(app, @ex_cbValueChanged,
true);
            app.ex_cb.Text = 'Execution';
            app.ex_cb.Position = [6 217 75 22];

            % Create MarkLabel
            app.MarkLabel = uilabel(app.LeftPanel);
            app.MarkLabel.Position = [94 280 46 22];
            app.MarkLabel.Text = 'Mark %';

            % Create ChannelselectionSwitchLabel
            app.ChannelselectionSwitchLabel = uilabel(app.LeftPanel);
            app.ChannelselectionSwitchLabel.HorizontalAlignment = 'center';
            app.ChannelselectionSwitchLabel.Position = [10 163 101 22];
            app.ChannelselectionSwitchLabel.Text = 'Channel selection';

            % Create ChannelselectionSwitch
            app.ChannelselectionSwitch = uiswitch(app.LeftPanel, 'slider');
            app.ChannelselectionSwitch.Items = {'Auto', 'Manual'};
            app.ChannelselectionSwitch.Enable = 'off';
            app.ChannelselectionSwitch.Position = [45 144 45 20];
            app.ChannelselectionSwitch.Value = 'Auto';

            % Create OpenxlsreportsButton
            app.OpenxlsreportsButton = uibutton(app.LeftPanel, 'push');
            app.OpenxlsreportsButton.ButtonPushedFcn = createCallbackFcn(app,
@OpenxlsreportsButtonPushed, true);
            app.OpenxlsreportsButton.Enable = 'off';
```

```
            app.OpenxlsreportsButton.Position = [13 18 120 22];
            app.OpenxlsreportsButton.Text = 'Open xls reports';

            % Create symm_mark_box
            app.symm_mark_box = uieditfield(app.LeftPanel, 'numeric');
            app.symm_mark_box.Limits = [0 100];
            app.symm_mark_box.ValueDisplayFormat = '%.0f';
            app.symm_mark_box.HorizontalAlignment = 'center';
            app.symm_mark_box.Position = [94 261 45 18];
            app.symm_mark_box.Value = 45;

            % Create dam_mark_box
            app.dam_mark_box = uieditfield(app.LeftPanel, 'numeric');
            app.dam_mark_box.Limits = [0 100];
            app.dam_mark_box.ValueDisplayFormat = '%.0f';
            app.dam_mark_box.HorizontalAlignment = 'center';
            app.dam_mark_box.Position = [94 240 45 18];
            app.dam_mark_box.Value = 55;

            % Create ex_mark_box
            app.ex_mark_box = uieditfield(app.LeftPanel, 'numeric');
            app.ex_mark_box.Limits = [0 100];
            app.ex_mark_box.ValueDisplayFormat = '%.0f';
            app.ex_mark_box.Editable = 'off';
            app.ex_mark_box.HorizontalAlignment = 'center';
            app.ex_mark_box.Position = [94 219 45 18];

            % Create Reset
            app.Reset = uibutton(app.LeftPanel, 'push');
            app.Reset.ButtonPushedFcn = createCallbackFcn(app, @ResetButtonPushed,
true);
            app.Reset.Position = [13 188 120 22];
            app.Reset.Text = 'Reset default';

            % Create CenterPanel
            app.CenterPanel = uipanel(app.GridLayout);
            app.CenterPanel.Layout.Row = 1;
            app.CenterPanel.Layout.Column = 2;

            % Create UITable
            app.UITable = uitable(app.CenterPanel);
            app.UITable.ColumnName = '';
            app.UITable.ColumnWidth = {'fit'};
            app.UITable.RowName = {};
            app.UITable.DisplayDataChangedFcn = createCallbackFcn(app,
@UITableDisplayDataChanged, true);
            app.UITable.Visible = 'off';
            app.UITable.Position = [1 7 686 355];

            % Create TextArea
            app.TextArea = uitextarea(app.CenterPanel);
            app.TextArea.Editable = 'off';
            app.TextArea.WordWrap = 'off';
            app.TextArea.FontSize = 20;
            app.TextArea.BackgroundColor = [0.9412 0.9412 0.9412];
            app.TextArea.Visible = 'off';
            app.TextArea.Position = [6 6 680 351];

            % Create ListBox
```

```
            app.ListBox = uilistbox(app.CenterPanel);
            app.ListBox.Items = {'', ''};
            app.ListBox.Multiselect = 'on';
            app.ListBox.Visible = 'off';
            app.ListBox.Position = [6 6 680 348];
            app.ListBox.Value = {''};

            % Create RightPanel
            app.RightPanel = uipanel(app.GridLayout);
            app.RightPanel.Layout.Row = 1;
            app.RightPanel.Layout.Column = 3;

            % Show the figure after all components are created
            app.GoTAAppUIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = GP_App_exported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.GoTAAppUIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.GoTAAppUIFigure)
        end
    end
end
```

10.2. Main function code

```
function [summary, error, reports_dir, graphs_dir]=main_GP(app)

    clearvars -except app
    clc, format long

    %setting error parameter as feedback
    error=1;
    summary=0;
    reports_dir=0;
    graphs_dir=0;
    % GETTING THE CAR DIRECTORY
    app.statusLabel.Text = 'Loading folder';
    app.statusLamp.Color = [0 1 1];
    drawnow;
    dirn = uigetdir(pwd, 'Select a folder'); %selection of directory
    if dirn==0
        error=104;
        figure(app.GoTAAppUIFigure)
    else
        %get the app in foreground and changing label and lamp
        figure(app.GoTAAppUIFigure)
        app.statusLamp.Color = [1 0.41 0];
        app.statusLabel.HorizontalAlignment = 'center';
        app.statusLabel.Text = 'Loading...';
        drawnow;

        dirname = dir(dirn); %get all directories from selected folder
        dfolders = dirname([dirname(:).isdir]); %save just the names of folders as cells
        dfolders = dfolders(~ismember({dfolders(:).name},{'.','..'})); %delete
unnecessary '.' and '..' folders
        [sfold, ~]=size(dfolders); %count folders
        names_dir=strings(sfold); %create a string array for folders name
        paths_dir=strings(sfold); %create a string array for folders path

        %Get the path for templates

        if isdeployed %if app is used select the installation directory
            [~, result] = system('path');
            working_dir = char(regexpi(result, 'Path=(.*?);', 'tokens', 'once'));
        else %else use the working directory
            working_dir = pwd;
        end

        %DELETE OLD REPORT AND GRAPHS FOLDERS IF EXIST AND MAKE NEW ONES

        template_dir=strcat(working_dir, "\templates");
%creating the names/paths for the new folders and reports
        summ_templ=strcat(template_dir, "\GP Summary.xlsx");
        reports_dir=strcat(dirn, "\Reports");
        graphs_dir=strcat(dirn, "\Plots");
        skip=0;
        tr_index=0;
        fold1=0;
        fold2=0;
        fold3=0;
```

```
        %Assign mark percentage values inserted from the user
        mk_s=app.symm_mark_box.Value/100;
        mk_d=app.dam_mark_box.Value/100;
        mk_e=app.ex_mark_box.Value/100;
        box_s=app.symm_cb.Value;
        box_d=app.dam_cb.Value;
        box_e=app.ex_cb.Value;
        if and(box_s,box_d)
            mk_s_lng=0.45;
            mk_d_lng=0.55;
        elseif box_s==1
            mk_s_lng=1;
            mk_d_lng=0;
        elseif box_d==1
            mk_s_lng=0;
            mk_d_lng=1;
        end

        if box_s+box_d+box_e==1
            app.UITable.ColumnWidth = {'fit',40,40,40,40,40,40,'fit'};
        elseif box_s+box_d+box_e==2
            app.UITable.ColumnWidth = {'fit',40,40,40,40,40,40,40,40,40,'fit'};
        elseif box_s+box_d+box_e==3
            app.UITable.ColumnWidth = {'fit',40,40,40,40,40,40,40,40,40,40,40,40,'fit'};
        end

        for h=1:sfold %for loop to make sure there are not old non empty directories for
graphs and reports
            paths_dir(h)=strcat(dfolders(h).folder,"\",dfolders(h).name); %saves the
path of each folder in "paths_dir"
            names_dir(h)=strcat(dfolders(h).name);  %saves the name of each folder in
"names_dir"
            if
and(or(contains(names_dir(h),"execution"),contains(names_dir(h),"damage")),skip==0)
%when for loop encounters execution or damage saves it in order to get the name of the
tests
                dfiles=paths_dir(h);
                skip=1;
            elseif paths_dir(h)==reports_dir
                rmdir(reports_dir,'s'); %if reports dir exist, it removes it
            elseif paths_dir(h)==graphs_dir
                rmdir(graphs_dir,'s'); %if graphs dir exist, it removes it
            end
            %if conditions to check if the three folders are present to compute
            %the complete analysis and if the user selected the respective box
            if or(and(contains(names_dir(h),"levelcrossing"),box_s==1),box_s==0)
                fold1=1;
            end
            if or(and(contains(names_dir(h),"damage"),box_d==1),box_d==0)
                fold2=1;
            end
            if or(and(contains(names_dir(h),"execution"),box_e==1),box_e==0)
                fold3=1;
            end
            if contains(names_dir(h),"levelcrossing")
                tr_index=1;
            end
        end
        mkdir(reports_dir); %makes new reports dir
```

```
        mkdir(graphs_dir); %makes new graphs dir

        %check the presence of the level crossing, damage and execution folders
        if (fold1==1&&fold2==1&&fold3==1)

            %IMPORT THE LIST OF TRACKS THAT CAN BE ANALIZED
            track_file=strcat(template_dir, "\tracks.txt");
            track_txt1 = importdata(track_file);
            track_txt = string(track_txt1);
            [n_tr_txt, ~]=size(track_txt);

            if skip==1
                %GET THE FILES' NAMES FROM THE SAVED EXECUTION OR DAMAGE FOLDER
                files = dir(fullfile(dfiles, '*.csv'));
                f=string({files.name});
                f=f';

                %CHECK WHICH TRACKS HAVE THREE TRIES NECESSARY FOR THE GP EVALUATION

                [n_s3t, ~]=size(f); %gets the number of test to check by counting files
in damage or execution
    %             n_tries=3;  %number of tries needed to evaluate the GP
                htr=1;  %counter of track that have enough tries to evaluate GP
                n_tries_tr=zeros(n_tr_txt, 1);  %array that counts the tries for each
track
                test_tr=strings(n_tr_txt,1);    %array with tracks to be tested
                pos_test_tr=zeros(n_tr_txt,1);  %array of positions for ordering

                %double for loop 'hh' runs through the list of tracks that the script
                %can evaluate and 'hhh' trough the tests names
                for hh=1:n_tr_txt
                    n_tries=0;
                    for hhh=1:n_s3t
                        if contains(f(hhh), track_txt(hh))
%check if the file 'f(hhh)' contains in the name 'track_txt(hh)'
                            n_tries=n_tries+1;    %counts the tries of track
'track_txt(hh)'
                            if n_tries==1
                                pos=hhh;
                            end
                        end
                    end
                    %if the tracks hh has enough tries, saves it in 'test_tr' and
                    %counts it
                    if n_tries>=1
                        test_tr(htr)=track_txt(hh);
                        pos_test_tr(htr)=pos;
                        n_tries_tr(htr)=n_tries;
                        htr=htr+1;
                    end
                end
                %cuts the arrays removing empty spots
                test_tr=test_tr(1:htr-1);
                pos_test_tr=pos_test_tr(1:htr-1);
                n_tries_tr=n_tries_tr(1:htr-1);

            elseif and(skip==0,tr_index==1)

            end
```

```
            %RUNNING THE THREE SUB FUNCTIONS FOR SYMMETRY, DAMAGE AND EXECUTION

            for h=1:sfold
                bfold=char(convertStringsToChars(names_dir(h)));
                if and(contains(bfold,"damage"),box_d==1)
                    d = paths_dir(h);
                    [track_names, mark, error]=b_average_damage(d, template_dir,
reports_dir, test_tr, error, app, n_tries_tr);
                    [n_tr, ~]=size(track_names);    %damage subfunction also gives a
confirmation of the number of tracks to be analysed
                    summ=zeros(n_tr,12);    %creates the summary mark matrix
                    summ(:,2:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the
damage marks into summ matrix
                    mark=0;

                elseif and(contains(bfold,"levelcrossing"),box_s==1)
                    d = paths_dir(h);
                    [mark, error]=a_symmetry(d, template_dir, reports_dir, test_tr,
graphs_dir, error, app, n_tries_tr);
                    summ(:,1:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the
symmetry marks into summ matrix
                    mark=0;

                elseif and(contains(bfold,"execution"),box_e==1)
                    d = paths_dir(h);
                    [mark, error]=c_execution(d, template_dir, reports_dir, test_tr,
graphs_dir, error, app, n_tries_tr);
                    summ(:,3:4:12)=[mark(:,1), mark(:,2), mark(:,3)];   %copies the
execution marks into summ matrix
                    mark=0;

                end
            end

            %SUM THE POINTS OF EACH PARAMETER AND CHOOSING THE GP TRY

            GP=strings(n_tr,1);
            for i=1:n_tr
                    for ii=0:n_tries_tr(i)-1
                        if (mk_s>0&&mk_d>0&&mk_e>0)
                            check_mark=[summ(i,ii*4+1), summ(i,ii*4+2), summ(i,ii*4+3)];
%copies the marks of the paramewters into a temporary array
                        elseif (mk_s>0&&mk_d>0)
                            check_mark=[summ(i,ii*4+1), summ(i,ii*4+2)];
                        elseif (mk_d>0&&mk_e>0)
                            check_mark=[summ(i,ii*4+2), summ(i,ii*4+3)];
                        elseif mk_d>0
                            check_mark=summ(i,ii*4+2);
                        end
                        if and(check_mark>0, check_mark<=10)
%checks if the marks are valid (between 0 and 10)

summ(i,ii*4+4)=(mk_s*summ(i,ii*4+1)+mk_d*summ(i,ii*4+2)+mk_e*summ(i,ii*4+3));
%weighted average of the three marks
                        end
                    end
                %checks the higher mark between the weighted averages and
                %assigns the GP try
```

```
                [track_mark, high_mark]=max(summ(i,4:4:12));
                if and(track_mark>0, track_mark<=10)
                    if high_mark==1
                        GP(i)=strcat("Try 1 ", f(pos_test_tr(i)));
                    elseif high_mark==2
                        GP(i)=strcat("Try 2 ", f(pos_test_tr(i)+1));
                    elseif high_mark==3
                        GP(i)=strcat("Try 3 ", f(pos_test_tr(i)+2));
                    end
                else %if no marks is between 0 and 10 there is no acceptable try
                    GP(i)="No acceptable try";
                end
            end

            %MAKING THE FINAL TABLE AND SUMMARY EXCEL FILE depeding on the
            %selected analysis performed

            summ=round(summ,2);
            summ=string(summ);
            %checks if langhe to change the zero into a '-'
            for i=1:n_tr
                if test_tr(i)=="LNG"
                        summ(i,3:4:11)='-'; %sets execution mark as null
                end
                %sets the empty tries to "-"
                if n_tries_tr(i)==1
                    summ(i,5:12)='-';
                elseif n_tries_tr(i)==2
                    summ(i,9:12)='-';
                end
            end
            copyfile(summ_templ, reports_dir)
            summ_file=strcat(reports_dir, "\GP Summary.xlsx");
            if (box_s&&box_d&&box_e)==1
                summary=table(track_names, summ(:,1), summ(:,2), summ(:,3), summ(:,4),
summ(:,5), summ(:,6), summ(:,7), summ(:,8), summ(:,9), summ(:,10), summ(:,11),
summ(:,12), GP(:));
                summary.Properties.VariableNames = {'Track names', 'S1', 'D1', 'E1',
'Fin1', 'S2', 'D2', 'E2', 'Fin2', 'S3', 'D3', 'E3', 'Fin3', 'Golden Pass try'};
            elseif (box_s&&box_d)==1
                summary=table(track_names, summ(:,1), summ(:,2), summ(:,4), summ(:,5),
summ(:,6), summ(:,8), summ(:,9), summ(:,10), summ(:,12), GP(:));
                summary.Properties.VariableNames = {'Track names', 'S1', 'D1', 'Fin1',
'S2', 'D2', 'Fin2', 'S3', 'D3', 'Fin3', 'Golden Pass try'};
            elseif (box_d&&box_e)==1
                summary=table(track_names, summ(:,2), summ(:,3), summ(:,4), summ(:,6),
summ(:,7), summ(:,8), summ(:,10), summ(:,11), summ(:,12), GP(:));
                summary.Properties.VariableNames = {'Track names', 'D1', 'E1', 'Fin1',
'D2', 'E2', 'Fin2', 'D3', 'E3', 'Fin3', 'Golden Pass try'};
            elseif box_d==1
                summary=table(track_names, summ(:,2), summ(:,4), summ(:,6), summ(:,8),
summ(:,10), summ(:,12), GP(:));
                summary.Properties.VariableNames = {'Track names', 'D1', 'Fin1', 'D2',
'Fin2', 'D3', 'Fin3', 'Golden Pass try'};
            end
            writetable(summary,summ_file,'sheet','GP');
            error=0;

        elseif fold1==0
```

```
            error=101; %Missing level crossing folder
        elseif fold2==0
            error=102; %Missing damage folder
        elseif fold3==0
            error=103; %Missing execution folder
        end
    end
end
```

10.3. Symmetry function code

```
function [mark_symm, error]=a_symmetry(d, template_dir, reports_dir, test_tr,
graphs_dir, error, app, n_tries_tr)

    %CHECKING THAT THE INPUTS HAVE THE RIGHT FORMAT

    arguments
        d (:,1) string
        template_dir (1,1) string
        reports_dir (1,1) string
        test_tr (:,1) string
        graphs_dir (1,1) string
        error double
        app
        n_tries_tr
    end

    clearvars -except names h sfold bfold d template_dir mark reports_dir test_tr
graphs_dir error app n_tries_tr
    format long
    clc
    error=2;

    app.statusLabel.HorizontalAlignment = 'left';
    app.statusLabel.Text = 'Symmetry 0%';

%     d= uigetdir(pwd, 'Select symmetry folder');
%     template_dir= strcat(pwd, "\templates");
%     reports_dir= strcat(d, "\Reports");
%     graphs_dir=strcat(d, "\Plots");
%     test_tr=["PABA","PORF-PROC","OB-PROC","PLX","MCISA","AV","POT","OL-DS","RAL","TR-
BU"];

    %IMPORTING SYMMETRY FILES' NAMES, CONVERTING INTO STRING AND CREATING
    %THE BLANK SYMMETRY REPORT

    files = dir(fullfile(d, '*.csv'));  %imports the .csv names
    f=strcat(d,'\',{files.name});       %makes an array of the files' full paths
    n_files=length(f);  %number of files

    %makes an array with simply the file names
    f_names=string({files.name});
    for ii=1:n_files
        f_names_prov=char(f_names(ii));
        f_names(ii)=f_names_prov(1:end-4);  %removes the characters '.csv' in order to
use the names for the graphs
    end

    %copies the symmetry report template in the new report folder
    symm_templ=strcat(template_dir, "\reportsymmetry.xlsx");
    copyfile(symm_templ, reports_dir)
    symm_file=strcat(reports_dir, "\reportsymmetry.xlsx");

    %DELETING THE OLD SYMM GRAPHS DIRECTORY IF PRESENT AND MAKING THE NEW ONE
    plotsdir_symm=strcat(graphs_dir,'\plots_symm'); %sets path for symm graphs folder
    dirname_symm = dir(d);  %saves attributes of d in dirname_symm
    dfolders_symm = dirname_symm([dirname_symm(:).isdir]);  %gets the folders in the
```

```
level crossing directory
    dfolders_symm = dfolders_symm(~ismember({dfolders_symm(:).name},{'.','..'}));
%removes the useless folders
    [sfold_symm, ~]=size(dfolders_symm);    %gets the number of folders
    if sfold_symm~=0
        names_symm=strings(sfold_symm,1);
        for h=1:sfold_symm
            names_symm(h)=strcat(dfolders_symm(h).folder,"\",dfolders_symm(h).name);
%saves directories' names into a string array
        end
        %checks if any graph folder is already present and deletes it
        if contains(names_symm,"plots_symm")
            rmdir(plotsdir_symm,'s');
        end
    end
    mkdir(plotsdir_symm);   %makes a new graph folder
    n_tries=3; %max number of tries

    %CHECKING THE TRACKS AVAILABLE IN THE SYMMETRY FOLDER

    sheetname=strings(n_files,1);
    n_tracks=0;
    checkPABA=0;
    checkAPES=0;
    checkPORFPROC=0;
    checkOBPROC=0;
    checkLNG=0;
    checkPLX=0;
    checkMCISA=0;
    checkAV=0;
    checkOLCLAS=0;
    checkOLDS=0;
    checkOLC50=0;
    checkTRBU=0;
    checkRAL=0;
    checkPOT=0;
    for i=1:n_files
        if checkPABA==0 %for each track
            if sum(and(contains(test_tr,"PABA"),contains(f(i),"PABA")))==1  %is this
track present in the list to be analyzed from the main? is the track in the list of
files for level crossing?
                n_tracks=n_tracks+1;    %if yes increase the number of track for
symmetry analysis
                sheetname(n_tracks)="PABA"; %save the name of the track in the array for
sheetnames of the report
                checkPABA=1;    %skip this track in the next iteration
            end
        end
        if checkAPES==0
            if sum(and(contains(test_tr,"APES"),contains(f(i),"APES")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="APES";
                checkAPES=1;
            end
        end
        if checkPORFPROC==0
            if sum(and(contains(test_tr,"PORF-PROC"),contains(f(i),"PORF-PROC")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="PORF-PROC";
```

```
            checkPORFPROC=1;
        end
    end
    if checkOBPROC==0
        if sum(and(contains(test_tr,"OB-PROC"),contains(f(i),"OB-PROC")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="OB-PROC";
            checkOBPROC=1;
        end
    end
    if checkLNG==0
        if sum(and(contains(test_tr,"LNG"),contains(f(i),"LNG")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="LNG";
            checkLNG=1;
        end
    end
    if checkPLX==0
        if sum(and(contains(test_tr,"PLX"),contains(f(i),"PLX")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="PLX";
            checkPLX=1;
        end
    end
    if checkMCISA==0
        if sum(and(contains(test_tr,"MCISA"),contains(f(i),"MCISA")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="MCISA";
            checkMCISA=1;
        end
    end
    if checkAV==0
        if sum(and(contains(test_tr,"AV"),contains(f(i),"AV")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="AV";
            checkAV=1;
        end
    end
    if checkOLCLAS==0
        if sum(and(contains(test_tr,"OLC-LAS"),contains(f(i),"OLC-LAS")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="OLC-LAS";
            checkOLCLAS=1;
        end
    end
    if checkOLDS==0
        if sum(and(contains(test_tr,"OL-DS"),contains(f(i),"OL-DS")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="OL-DS";
            checkOLDS=1;
        end
    end
    if checkOLC50==0
        if sum(and(contains(test_tr,"OLC50"),contains(f(i),"OLC50")))==1
            n_tracks=n_tracks+1;
            sheetname(n_tracks)="OLC50";
            checkOLC50=1;
        end
    end
```

```
        if checkTRBU==0
            if sum(and(contains(test_tr,"TR-BU"),contains(f(i),"TR-BU")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="TR-BU";
                checkTRBU=1;
            end
        end
        if checkRAL==0
            if sum(and(contains(test_tr,"RAL"),contains(f(i),"RAL")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="RAL";
                checkRAL=1;
            end
        end
        if checkPOT==0
            if sum(and(contains(test_tr,"POT"),contains(f(i),"POT")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="POT";
                checkPOT=1;
            end
        end
    end

    %COUNTING THE TOTAL NUMBER OF CHANNELS FOR EACH TRACK

    sheetname=sheetname(1:n_tracks);    %cuts the non empty entries of the sheetname
array
    n_ch=zeros(n_tracks,1);
    %double for that counts the files for each track
    for i=1:n_tracks
        for ii=1:n_files
            if contains(f(ii),sheetname(i))
                n_ch(i)=n_ch(i)+1;
            end
        end
    end
    n_ch=n_ch./n_tries_tr;  %divides by tries to have the channel number

    %START OF THE WHILE LOOP FOR SYMMETRY CHECK

    Ch_n=strings(n_files,1);
    ab=1;   %"ab" runs through all the files in the levelcrossing folder
    as=1;   %"as" is the index for the track that is being analyzed through the three
tries
    skip=zeros(n_files,1);  %to save the channels that must be skipped because already
ordered
    Ch_n_ord=strings(n_files,1);    %to save the ordered channels' names
    f_ord=strings(n_files,1);       %to save the ordered files' names
    pt_track=zeros(n_tracks,n_tries);   %matrix for symmetry points of each track and
try
    t_pts=zeros(n_tracks,n_tries);      %position array for points of standard deviation
    t_ptx=zeros(n_tracks,n_tries);      %position array for points of maxima diff
    t_ptn=zeros(n_tracks,n_tries);      %position array for points of minima diff
    t_pti=zeros(n_tracks,n_tries);      %position array for points of integral diff
    graph_count=0;  %counter for percentage loading
    perc_symm=0;    %percentage value

    while ab<n_files
```

```
        %CHECK IF THE FILE ab IS OF A TRACK THAT HAS TO BE EVALUATED, IF
        %NOT, NEXT FILE

        if contains(f(ab),test_tr)

            ac=ab;  %index for channels of each single try
            c=1;
            mxR=zeros(n_ch(as),n_tries);
            mxL=zeros(n_ch(as),n_tries);
            mnR=zeros(n_ch(as),n_tries);
            mnL=zeros(n_ch(as),n_tries);
            stdev=zeros(n_ch(as),n_tries);
            tint=zeros(n_ch(as),n_tries);
            tint_diff=zeros(n_ch(as),n_tries);
            pt_stdev=zeros(n_ch(as),n_tries);
            pt_mxdiff=zeros(n_ch(as),n_tries);
            pt_mndiff=zeros(n_ch(as),n_tries);
            pt_int=zeros(n_ch(as),n_tries);
            mx_diff=zeros(n_ch(as),n_tries);
            mn_diff=zeros(n_ch(as),n_tries);
            int_perc=zeros(n_ch(as),n_tries);

            %WHILE LOOP FOR EVERY CHANNEL OF THE THREE TRIES OF A TRACK


            while ac<ab+n_ch(as)*n_tries_tr(as)-1
                %"for" that imports the channel names
                for i=ac:(ac+n_ch(as)-1)
                    name = importdata(f{i});
                    Ch_n(i) = string(name.textdata(9));
                end

                I=char(n_ch(as),1);
                R=strings(n_ch(as),1);
                %"for" that separates the initial letter from the rest of
                %the channel name for ordering purposes HIGHLY DEPENDS ON
                %NOMENCLATURE METHOD
                for i=ac:(ac+n_ch(as)-1)
                    a=convertStringsToChars(Ch_n(i));
                    I(i)=a(1);
                    R(i)=string(a(2:end));
                end

                Ch_n=Ch_n';
                i=ac;
                m=ac;
                k=ac+n_ch(as)-1;
                p=1;
                g=1;
                %channels to be analyzed
                symm_ch=["FX", "FY", "FZ", "AccX_Spindle", "AccY_Spindle",
"AccZ_Spindle", "AccX_EngineMount", "AccY_EngineMount", "AccZ_EngineMount", "AccX_Body",
"AccY_Body", "AccZ_Body", "Disp_ShockAbs"];

                %WHILE LOOP FOR EVERY CHANNEL OF *ONE* OF THE THREE TRIES OF A TRACK

                while and(i<=ac+n_ch(as)-1,m<k)
                    check=0;
```

```
                    %ORDERING LOOP IN COUPLES AND SELECTION OF CHANNELS TO
                    %BE ANALYZED
                    for j=i+1:ac+n_ch(as)-1
                        if ismember(i,skip) %if "i" in the skip array, next "i"
                            check=1;
                            break
                        end
                        control=contains(Ch_n(i),symm_ch);  %checks if the channel has
to be analyzed
                        if and((I(i) == 'L' || I(i)=='R'),control)  %according to
nomenclature, the channel should start with L (left) or R (right)
                            if R(i)==R(j)   %checks the "i" and "j" channel names apart
from the first letter
                                check=1;
                                skip(p)=j;
                                p=p+1;
                                g=g+2;
                                %if loop orders in couples, left channel
                                %first and right channel second
                                if I(i)=='L'
                                    Ch_n_ord(m)=Ch_n(i);
                                    Ch_n_ord(m+1)=Ch_n(j);
                                    f_ord(m)=f(i);
                                    f_ord(m+1)=f(j);
                                    m=m+2;
                                else
                                    Ch_n_ord(m)=Ch_n(j);
                                    Ch_n_ord(m+1)=Ch_n(i);
                                    f_ord(m)=f(j);
                                    f_ord(m+1)=f(i);
                                    m=m+2;
                                end
                            end
                        else %if the channel "i" does not start with L or R
                             %or is not in the list of channels to be analyzed,
                             %ends up at the end of the ordered list
                            Ch_n_ord(k)=Ch_n(i);
                            f_ord(k)=f(i);
                            k=k-1;
                            check=1;
                        end
                        %check==1 next "i"
                        if check==1
                            break
                        end
                    end
                    if check~=1 %if "i" start with L or R but does not have
                                %a match, goes at the end of the ordered list
                        Ch_n_ord(k)=Ch_n(i);
                        f_ord(k)=f(i);
                        k=k-1;
                    end
                    i=i+1;
                end
                n_couples=(g-1)/2;

                %FOR THE FIRST TRY AMONG THE THREE GETS A VECTOR WITH THE
                %ORDERED CHANNELS' NAMES
```

```
                if c==1 %c checks that is the first try
                    i=1;
                    j=1;
                    Channel_name=strings(n_ch(as),1);

                    while i<g   %while copies the just the coupled channel names
                        a=convertStringsToChars(Ch_n_ord(i));
                        I(i)=a(1);
                        Channel_name(j)=string(a(2:end));
                        j=j+1;
                        i=i+2;
                    end
                end

                %WHILE LOOP FOR IMPORTING THE DATA AND EVALUATE SYMMETRY

                i=ac;
                j=1;
                div=1000;
                while i<m   %i up to m that is the number of coupled channels

                    %import of datas

                    ChanL=readtable(f_ord{i},'NumHeaderLines',14);
                    ChanL=ChanL{:,:};
                    accL=ChanL(:,1);    %acc is the array with accelerometer values
                    nL=ChanL(:,2);      %n is the array with number of level crossing
repetitions

                    ChanR=readtable(f_ord{i+1},'NumHeaderLines',14);
                    ChanR=ChanR{:,:};
                    accR=ChanR(:,1);
                    nR=ChanR(:,2);
                    %for values measured on the Y axis the accelerometers
                    %values for the right side change sign
                    ctY=or(contains(Ch_n_ord(i),"FY"),contains(Ch_n_ord(i),"AccY"));
                    %for simplicity the values are copied into x and y
                    %arrays named for the cartesian coordinates they will
                    %be shown in the graphs
                    if(ctY==1)
                        xR=-1*accR;
                    else
                        xR=accR;
                    end
                    xL=accL;
                    yL=nL;
                    yR=nR;

                    %SPLINE EVALUATION, SUBTRACTION OF THE TWO CHANNELS AND
                    %SELECTION OF MAXIMA AND MINIMA

                    pR=spline(xR,yR);
                    pL=spline(xL,yL);
                    x=linspace(max(min(xR),min(xL)),min(max(xR),max(xL)),div);
                    yL1=ppval(pL,x);
                    yR1=ppval(pR,x);
                    ydiff=yL1-yR1;

                    mxR(j,c)=max(accR);
```

```
                    mxL(j,c)=max(accL);
                    mnR(j,c)=min(accR);
                    mnL(j,c)=min(accL);

                    %GRAPH CREATION AND SAVING

                    title_symm=strcat(f_names(i),"_", Channel_name(j), " symm");
                    fig = figure('visible','off');
                    plot(x,yL1,'--','color','r');
                    hold on;
                    plot(x,yR1,'--','color','b');
                    hold on;
                    plot(x,ydiff,'-','color','g','LineWidth',1);
                    title(title_symm,'Interpreter','none');
                    ylabel('level crossing divisions');
                    xlabel('acceleration [m/s^2]');
                    legend('Left','Right', 'Difference','location','best');
                    hold off;
                    figname=strcat(title_symm, '.fig');
                    savefig(fig, fullfile(plotsdir_symm, figname));
                    close (fig)
                    %updates the loading percentage
                    graph_count=graph_count+1;

perc_symm=round(graph_count*100/(n_couples*n_tracks*max(n_tries_tr)),0);
                    percentage_status=char(strcat("Symmetry ", string(perc_symm), "%"));
                    app.statusLabel.HorizontalAlignment = 'left';
                    app.statusLabel.Text = percentage_status;

                    %EVALUATION OF STANDARD DEVIATION AND INTEGRALS
                    int_diff=0;
                    int=0;
                    %for loop evaluates the absolute value of the integral
                    %of the difference spline
                    for e=1:div-1
                        int_diff=int_diff+abs(trapz(ydiff(e:e+1)));
                        int=int+abs(trapz(yL1(e:e+1)));
                    end

                    %save the values of integral of difference spline and
                    %the differences between maxima and minima into a three
                    %coloumns matrix in order to compare the results among
                    %the three tries
                    stdev(j,c)=std(ydiff);
                    tint_diff(j,c)=int_diff;
                    tint(j,c)=int;
                    mx_diff(j,c)=abs(mxR(j,c)-mxL(j,c))/max(mxR(j,c),mxL(j,c));
                    mn_diff(j,c)=abs(mnR(j,c)-
mnL(j,c))/max(abs(mnR(j,c)),abs(mnL(j,c)));
                    int_perc(j,c)=tint_diff(j,c)/tint(j,c);
                    j=j+1;
                    i=i+2;
                end
                c=c+1;
                ac=ac+n_ch(as);
            end

            %FOR LOOP FOR ASSIGNING SCORES
            %t is among 1 and 3 the position in the arrays of the values
```

```
            %and selects the try that has 3,2 or 1 points
            for pt=1:j-1
                [~, t]=max(stdev(pt,:));
                pt_stdev(pt,t)=1;
                [~, t]=min(stdev(pt,:));
                pt_stdev(pt,t)=3;
                [~, t]=min(pt_stdev(pt,:));
                pt_stdev(pt,t)=2;

                [~, t]=max(mx_diff(pt,:));
                pt_mxdiff(pt,t)=1;
                [~, t]=min(mx_diff(pt,:));
                pt_mxdiff(pt,t)=3;
                [~, t]=min(pt_mxdiff(pt,:));
                pt_mxdiff(pt,t)=2;

                [~, t]=max(mn_diff(pt,:));
                pt_mndiff(pt,t)=1;
                [~, t]=min(mn_diff(pt,:));
                pt_mndiff(pt,t)=3;
                [~, t]=min(pt_mndiff(pt,:));
                pt_mndiff(pt,t)=2;

                [~, t]=max(int_perc(pt,:));
                pt_int(pt,t)=1;
                [~, t]=min(int_perc(pt,:));
                pt_int(pt,t)=3;
                [~, t]=min(pt_int(pt,:));
                pt_int(pt,t)=2;
            end

            %CREATING THE SUMMARY TABLE FOR EACH TRACK AND SAVING IN THE
            %RESPECTIVE EXCEL REPORT SHEET

            n_cpl=1:1:n_couples;
            n_cpl=n_cpl';
            %sum of the points for each parameter (standard dev, integral
            %difference, max and min difference) among the evaluated channels
            if n_tries_tr(as)==1
                t_pts(as,1)=sum(pt_stdev(:,1));
                t_ptx(as,1)=sum(pt_mxdiff(:,1));
                t_ptn(as,1)=sum(pt_mndiff(:,1));
                t_pti(as,1)=sum(pt_int(:,1));
            %weighted average of the points to calculate the mark for each
            %try of the track

pt_track(as,1)=t_pts(as,1)*0.45+t_ptx(as,1)*0.075+t_ptn(as,1)*0.075+t_pti(as,1)*0.4;
            elseif n_tries_tr(as)==2
                t_pts(as,1:2)=[sum(pt_stdev(:,1)), sum(pt_stdev(:,2))];
                t_ptx(as,1:2)=[sum(pt_mxdiff(:,1)), sum(pt_mxdiff(:,2))];
                t_ptn(as,1:2)=[sum(pt_mndiff(:,1)), sum(pt_mndiff(:,2))];
                t_pti(as,1:2)=[sum(pt_int(:,1)), sum(pt_int(:,2))];

pt_track(as,1:2)=[t_pts(as,1)*0.45+t_ptx(as,1)*0.075+t_ptn(as,1)*0.075+t_pti(as,1)*0.4,
t_pts(as,2)*0.45+t_ptx(as,2)*0.075+t_ptn(as,2)*0.075+t_pti(as,2)*0.4];
            elseif n_tries_tr(as)==3
                t_pts(as,:)=[sum(pt_stdev(:,1)), sum(pt_stdev(:,2)),
sum(pt_stdev(:,3))];
                t_ptx(as,:)=[sum(pt_mxdiff(:,1)), sum(pt_mxdiff(:,2)),
```

```
sum(pt_mxdiff(:,3))];
                t_ptn(as,:)=[sum(pt_mndiff(:,1)), sum(pt_mndiff(:,2)),
sum(pt_mndiff(:,3))];
                t_pti(as,:)=[sum(pt_int(:,1)), sum(pt_int(:,2)), sum(pt_int(:,3))];

pt_track(as,:)=[t_pts(as,1)*0.45+t_ptx(as,1)*0.075+t_ptn(as,1)*0.075+t_pti(as,1)*0.4,
t_pts(as,2)*0.45+t_ptx(as,2)*0.075+t_ptn(as,2)*0.075+t_pti(as,2)*0.4,
t_pts(as,3)*0.45+t_ptx(as,3)*0.075+t_ptn(as,3)*0.075+t_pti(as,3)*0.4];
            end
            %writing and saving of the summary for the track
            report=table(n_cpl,Channel_name(1:j-1),tint_diff(1:j-1,1),tint(1:j-
1,1),mxR(1:j-1,1),mxL(1:j-1,1),mnR(1:j-1,1),mnL(1:j-1,1),stdev(1:j-1,1),pt_stdev(1:j-
1,1),mx_diff(1:j-1,1),pt_mxdiff(1:j-1,1),mn_diff(1:j-1,1),pt_mndiff(1:j-
1,1),int_perc(1:j-1,1),pt_int(1:j-1,1),tint_diff(1:j-1,2),tint(1:j-1,2),mxR(1:j-
1,2),mxL(1:j-1,2),mnR(1:j-1,2),mnL(1:j-1,2),stdev(1:j-1,2),pt_stdev(1:j-
1,2),mx_diff(1:j-1,2),pt_mxdiff(1:j-1,2),mn_diff(1:j-1,2),pt_mndiff(1:j-
1,2),int_perc(1:j-1,2),pt_int(1:j-1,2),tint_diff(1:j-1,3),tint(1:j-1,3),mxR(1:j-
1,3),mxL(1:j-1,3),mnR(1:j-1,3),mnL(1:j-1,3),stdev(1:j-1,3),pt_stdev(1:j-
1,3),mx_diff(1:j-1,3),pt_mxdiff(1:j-1,3),mn_diff(1:j-1,3),pt_mndiff(1:j-
1,3),int_perc(1:j-1,3),pt_int(1:j-1,3));
            report.Properties.VariableNames = {'Couple n.','Couple name','Int of diff
1','Max integral 1','MaxR 1','MaxL 1','MinR 1','MinL 1','St dev 1','Ptsdev1','Max diff
1','Ptmx1','Min diff 1','Ptmn1','Int % 1','Ptint1','Int of diff 2','Max integral
2','MaxR 2','MaxL 2','MinR 2','MinL 2','St dev 2','Ptsdev2','Max diff 2','Ptmx2','Min
diff 2','Ptmn2','Int % 2','Ptint2','Int of diff 3','Max integral 3','MaxR 3','MaxL
3','MinR 3','MinL 3','St dev 3','Ptsdev3','Max diff 3','Ptmx3','Min diff 3','Ptmn3','Int
% 3','Ptint3'};
            writetable(report,symm_file,'sheet',sheetname(as));
            ab=ab+n_ch(as)*n_tries_tr(as);
            as=as+1;    %next track
        else    %if file ab is not in the tracks to be analyzed, next file
            ab=ab+1;
        end
    end

    %CREATING THE OVERALL SYMMETRY REPORT TABLE AND SAVING IN THE EXCEL
    %REPORT FILE

    track_names=sheetname;
    pt_tot(1)=3*n_couples;
    %final symmetry mark out of 10 of each try
    mark_symm(:,1:3)=[pt_track(:,1)*10/pt_tot(1), pt_track(:,2)*10/pt_tot(1),
pt_track(:,3)*10/pt_tot(1)];
    summ=[pt_track(:,1), mark_symm(:,1), pt_track(:,2), mark_symm(:,2), pt_track(:,3),
mark_symm(:,3)];
    %writing and saving of final summary table
    summary=table(track_names, summ(:,2), summ(:,4), summ(:,6));
    summary.Properties.VariableNames = {'Track names', 'Mark 1', 'Mark 2', 'Mark 3'};
    writetable(summary,symm_file,'sheet','Summary');
    error=1;
end
```

10.4. Average damage function code

```
function [track_names, mark_damage, error]=b_average_damage(d, template_dir,
reports_dir, test_tr, error, app, n_tries_tr)

    %CHECKING THAT THE INPUTS HAVE THE RIGHT FORMAT

    arguments
        d (:,1) string
        template_dir (1,1) string
        reports_dir (1,1) string
        test_tr (:,1) string
        error double
        app
        n_tries_tr
    end
    clearvars -except names h sfold bfold d template_dir mark reports_dir test_tr
graphs_dir error app n_tries_tr
    format long
    clc
    error=3;

    app.statusLabel.HorizontalAlignment = 'left';
    app.statusLabel.Text = 'Damage 0%';

%     d= uigetdir(pwd, 'Select damage folder');
%     template_dir= strcat(pwd, "\templates");
%     reports_dir= strcat(d, "\Reports");
%     test_tr=["PABA","PORF-PROC","OB-PROC","PLX","MCISA","AV","POT","OL-DS","RAL","TR-
BU"];

    %COPYING THE BLANK DAMAGE REPORT

    damage_templ=strcat(template_dir, "\reportdamage.xlsx");
    copyfile(damage_templ, reports_dir)
    damage_file=strcat(reports_dir, "\reportdamage.xlsx");

    %IMPORTING DAMAGE FILES' NAMES, CONVERTING INTO STRING
    files = dir(fullfile(d, '*.csv'));
    f=strcat(d,'\',{files.name});

    g=char(convertStringsToChars(f(1)));
    dmg=readtable(g);   %import first file to count channels
    n_files=length(f);
    n_tries=3;
    sz=size(dmg);
    n_ch_prov=sz(1);
    wM=n_tries*3+2; %width of M is 3 values for each try plus 2 extra columns
    M=zeros(n_ch_prov,wM);
    M_prov=zeros(n_ch_prov,wM);
    Dam_prov=strings(n_ch_prov,1);
    Dam=strings(n_ch_prov,1);
    Ch_name_prov=strings(n_ch_prov,1);
    Ch_name=strings(n_ch_prov,1);
    Ch_n_prov=zeros(n_ch_prov,1);
    Ch_n=zeros(n_ch_prov,1);
    sheetname=strings(n_files,1);
```

```
    %CHECKING THE TRACKS AVAILABLE IN THE DAMAGE FOLDER

    n_tracks=0;
    checkPABA=0;
    checkAPES=0;
    checkPORFPROC=0;
    checkOBPROC=0;
    checkLNG=0;
    checkPLX=0;
    checkMCISA=0;
    checkAV=0;
    checkOLCLAS=0;
    checkOLDS=0;
    checkOLC50=0;
    checkTRBU=0;
    checkRAL=0;
    checkPOT=0;

    for i=1:n_files
        if checkPABA==0     %for each track
            if sum(and(contains(test_tr,"PABA"),contains(f(i),"PABA")))==1  %is this
track present in the list to be analyzed from the main? is the track in the list of
files for level crossing?
                n_tracks=n_tracks+1;    %if yes increase the number of track for
symmetry analysis
                sheetname(n_tracks)="PABA"; %save the name of the track in the array for
sheetnames of the report
                checkPABA=1;            %skip this track in the next iteration
            end
        end
        if checkAPES==0
            if sum(and(contains(test_tr,"APES"),contains(f(i),"APES")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="APES";
                checkAPES=1;
            end
        end
        if checkPORFPROC==0
            if sum(and(contains(test_tr,"PORF-PROC"),contains(f(i),"PORF-PROC")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="PORF-PROC";
                checkPORFPROC=1;
            end
        end
        if checkOBPROC==0
            if sum(and(contains(test_tr,"OB-PROC"),contains(f(i),"OB-PROC")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="OB-PROC";
                checkOBPROC=1;
            end
        end
        if checkLNG==0
            if sum(and(contains(test_tr,"LNG"),contains(f(i),"LNG")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="LNG";
                checkLNG=1;
            end
        end
```

```
        if checkPLX==0
            if sum(and(contains(test_tr,"PLX"),contains(f(i),"PLX")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="PLX";
                checkPLX=1;
            end
        end
        if checkMCISA==0
            if sum(and(contains(test_tr,"MCISA"),contains(f(i),"MCISA")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="MCISA";
                checkMCISA=1;
            end
        end
        if checkAV==0
            if sum(and(contains(test_tr,"AV"),contains(f(i),"AV")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="AV";
                checkAV=1;
            end
        end
        if checkOLCLAS==0
            if sum(and(contains(test_tr,"OLC-LAS"),contains(f(i),"OLC-LAS")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="OLC-LAS";
                checkOLCLAS=1;
            end
        end
        if checkOLDS==0
            if sum(and(contains(test_tr,"OL-DS"),contains(f(i),"OL-DS")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="OL-DS";
                checkOLDS=1;
            end
        end
        if checkOLC50==0
            if sum(and(contains(test_tr,"OLC50"),contains(f(i),"OLC50")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="OLC50";
                checkOLC50=1;
            end
        end
        if checkTRBU==0
            if sum(and(contains(test_tr,"TR-BU"),contains(f(i),"TR-BU")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="TR-BU";
                checkTRBU=1;
            end
        end
        if checkRAL==0
            if sum(and(contains(test_tr,"RAL"),contains(f(i),"RAL")))==1
                n_tracks=n_tracks+1;
                sheetname(n_tracks)="RAL";
                checkRAL=1;
            end
        end
        if checkPOT==0
            if sum(and(contains(test_tr,"POT"),contains(f(i),"POT")))==1
                n_tracks=n_tracks+1;
```

```
                sheetname(n_tracks)="POT";
                checkPOT=1;
            end
        end
    end
    sheetname=sheetname(1:n_tracks);

    %FOR LOOP FOR EVALUATION OF THE AVERAGE DAMAGE

    dev=zeros(n_tracks,n_tries);
    damage=zeros(n_tracks,n_tries);
    pt_dev=zeros(n_tracks,n_tries);
    pt_damage=zeros(n_tracks,n_tries);
    pt_track=zeros(n_tracks,n_tries);
    t_pts=zeros(n_tracks,n_tries);
    t_ptd=zeros(n_tracks,n_tries);
    perc_count=0;
    perc_dam=0;

    %for loop to run through the tracks
    for m=1:n_tracks
        %channels to be analyzed
        dam_ch=["FX", "FY", "FZ", "AccX", "AccY", "AccZ", "Disp"];
        j=1;

        %FOR LOOP FOR EACH TRY

        for i=1:n_tries_tr(m)
            b=char(convertStringsToChars(f((m-1)*n_tries_tr(m)+i)));
            dmg1=readtable(b);  %import file
            %split the table in number of channel, name and damage value
            Ch_n_prov=table2array(dmg1(:,1));
            Ch_name_prov=table2array(dmg1(:,2));
            Dam_prov=string(table2array(dmg1(:,3)));
            [n_ch_prov, ~]=size(Ch_n_prov);
            jj=1;
            n_ch=0;

            %CHECK OF THE CHANNELS TO BE EVALUATED AND COUNTING

            for ii=1:n_ch_prov
                if contains(Ch_name_prov(ii), dam_ch)
                    Ch_n(jj)=Ch_n_prov(ii);
                    Ch_name(jj)=Ch_name_prov(ii);
                    Dam(jj,:)=Dam_prov(ii,:);
                    jj=jj+1;
                    n_ch=n_ch+1;
                end
            end
            M_prov(:,j)=Dam;
            j=j+3;
        end

        %cut of the Matrix and arrays to the actual number of channels
        M=M_prov(1:n_ch,:);
        Ch_n=Ch_n(1:n_ch);
        Ch_name=Ch_name(1:n_ch);

        %CALCULATION OF MEAN AND STANDARD DEVIATION FOR EACH CHANNEL
```

```
        for i=1:n_ch    %for each channel
            M(i,wM-1)=mean(M(i,1:3:7)); %mean of damage value
            for j=0:n_tries_tr(m)-1   %for each channel and for each try
                M(i,j*3+2)=M(i,j*3+1)/M(i,wM-1);    %damage over average damage
                M(i,j*3+3)=((M(i,j*3+1)/M(i,wM-1))-1)^2;    %standard deviation
                M(i,wM)=M(i,wM)+M(i,j*3+3);     %sum of standard deviation
            end
        end

        pt_stdev=zeros(n_ch,n_tries);
        pt_dam=zeros(n_ch,n_tries);

        %FOR LOOP FOR ASSIGNING SCORES
        %t is among 1 and 3 the position in the arrays of the values
        %and selects the try that has 3,2 or 1 points
        for pt=1:n_ch
            [~, t]=max(M(pt,3:3:9));
            pt_stdev(pt,t)=1;
            [~, t]=min(M(pt,3:3:9));
            pt_stdev(pt,t)=3;
            [~, t]=min(pt_stdev(pt,:));
            pt_stdev(pt,t)=2;

            [~, t]=max(abs(M(pt,2:3:8)-1));
            pt_dam(pt,t)=1;
            [~, t]=min(abs(M(pt,2:3:8)-1));
            pt_dam(pt,t)=3;
            [~, t]=min(pt_dam(pt,:));
            pt_dam(pt,t)=2;

            %updating loading percentage
            perc_count=perc_count+1;
            perc_dam=round(perc_count*100/(n_ch*n_tracks),0);
            percentage_status=char(strcat("Damage ", string(perc_dam), "%"));
            app.statusLabel.HorizontalAlignment = 'left';
            app.statusLabel.Text = percentage_status;
            drawnow;
        end

        if n_tries_tr(m)==1
        %sum of the points for standard dev and damage for each track
        t_pts(m,1)=sum(pt_stdev(:,1));
        t_ptd(m,1)=sum(pt_dam(:,1));
        %weigthed average to get the points of each track
        pt_track(m,1)=t_pts(m,1)*0.5+t_ptd(m,1)*0.5;
        elseif n_tries_tr(m)==2
        t_pts(m,1:2)=[sum(pt_stdev(:,1)), sum(pt_stdev(:,2))];
        t_ptd(m,1:2)=[sum(pt_dam(:,1)), sum(pt_dam(:,2))];
        pt_track(m,1:2)=[t_pts(m,1)*0.5+t_ptd(m,1)*0.5, t_pts(m,2)*0.5+t_ptd(m,2)*0.5];
        elseif n_tries_tr(m)==3
        t_pts(m,:)=[sum(pt_stdev(:,1)), sum(pt_stdev(:,2)), sum(pt_stdev(:,3))];
        t_ptd(m,:)=[sum(pt_dam(:,1)), sum(pt_dam(:,2)), sum(pt_dam(:,3))];
        pt_track(m,:)=[t_pts(m,1)*0.5+t_ptd(m,1)*0.5, t_pts(m,2)*0.5+t_ptd(m,2)*0.5,
t_pts(m,3)*0.5+t_ptd(m,3)*0.5];
        end
        %creating and saving of report table
        report=table(Ch_name, Ch_n, M(:,1), M(:,2), pt_dam(:,1), M(:,3), pt_stdev(:,1),
M(:,4), M(:,5), pt_dam(:,2), M(:,6), pt_stdev(:,2), M(:,7), M(:,8), pt_dam(:,3), M(:,9),
```

```
pt_stdev(:,3), M(:,10), M(:,11));
        report.Properties.VariableNames = {'Channel name', 'Chan N', 'Dam1', 'Dam1/Avg
Dam', 'Pt dam1', '(D1/A-1)²', 'Pt sdev1', 'Dam2', 'Dam2/Avg Dam', 'Pt dam2', '(D2/A-
1)²', 'Pt sdev2', 'Dam3', 'Dam3/Avg Dam', 'Pt dam3', '(D3/A-1)²', 'Pt sdev3', 'Avg
damage', 'Σ(D/A-1)²'};
        writetable(report,damage_file,'sheet',sheetname(m));
    end

    %MAKING THE OVERALL SUMMARY TABLE FOR DAMAGE AND SAVING IT

    track_names=sheetname;
    pt_tot(2)=3*n_ch;
    %calculates the mark of damage out of 10 for each track
    mark_damage(:,1:3)=[pt_track(:,1)*10/pt_tot(2), pt_track(:,2)*10/pt_tot(2),
pt_track(:,3)*10/pt_tot(2)];
    summ=[pt_track(:,1), mark_damage(:,1), pt_track(:,2), mark_damage(:,2),
pt_track(:,3), mark_damage(:,3)];
    summary=table(track_names, summ(:,2), summ(:,4), summ(:,6));
    summary.Properties.VariableNames = {'Track names', 'Mark 1', 'Mark 2', 'Mark 3'};
    writetable(summary,damage_file,'sheet','Summary');
    error=1;
end
```

10.5. Execution function code

```
function [mark_execution, error]=c_execution(d, template_dir, reports_dir, test_tr,
graphs_dir, error, app, n_tries_tr)

    %CHECKING THAT THE INPUTS HAVE THE RIGHT FORMAT

    arguments
        d (:,1) string
        template_dir (1,1) string
        reports_dir (1,1) string
        test_tr (:,1) string
        graphs_dir (1,1) string
        error double
        app
        n_tries_tr
    end
    clearvars -except names h sfold bfold d template_dir mark reports_dir test_tr
graphs_dir error app n_tries_tr
    format long
    clc
    error=4;

    app.statusLabel.HorizontalAlignment = 'left';
    app.statusLabel.Text = 'Execution 0%';

%     d= uigetdir(pwd, 'Select execution folder');
%     template_dir= strcat(pwd, "\templates");
%     reports_dir= strcat(d, "\Reports");
%     graphs_dir=strcat(d, "\Plots");
%     test_tr=["AV","LNG","MCISA","OB-PROC","OL-DS","OLC-LAS","PABA","PLX","PORF-
PROC","POT","RAL","TR-BU"];


    %COORDINATES SETS FOR EACH TRACK
    %these sets of coordinates are the important points of each track
    %indicating start, end and crucial speed points
    n=100000;
    lt_paba=[45.48385, 45.48247, 45.48231, 45.48181, 45.48043];
    lg_paba=[8.301329, 8.300705, 8.300628, 8.300398, 8.299769];

    lt_apes=[];
    lg_apes=[];

    lt_porfproc=[45.48037, 45.48388];
    lg_porfproc=[8.299657, 8.301170];

    lt_obproc=[45.48046, 45.48203, 45.48371];
    lg_obproc=[8.299628, 8.300336, 8.301106];

    lt_plx=[45.47551, 45.47896, 45.47941, 45.47999, 45.48055, 45.48327, 45.48391,
45.48324, 45.48387, 45.47791]; %#ok<NASGU>
    lg_plx=[8.280850, 8.286669, 8.287610, 8.288610, 8.289512, 8.293789, 8.293245,
8.292613, 8.293303, 8.281163];

    lt_mcisa=[45.47785, 45.48055];
    lg_mcisa=[8.304417, 8.305412];
```

```
lt_olclas=[45.48405, 45.48073, 45.48046, 45.48391];
lg_olclas=[8.301132, 8.299650, 8.299592, 8.301128];

lt_olds=[45.48395, 45.48297, 45.48297, 45.48395];
lg_olds=[8.301150, 8.300674, 8.300360, 8.301141];

lt_trbu=[45.47915, 45.48007, 45.48200];
lg_trbu=[8.298965, 8.299380, 8.300588];

lt_ral=[45.47838, 45.47840, 45.47856, 45.47840];
lg_ral=[8.298661, 8.298891, 8.298797, 8.298877];

lt_pot=[45.47979, 45.47887];
lg_pot=[8.299331, 8.298964];

%IMPORTING FILES' NAMES, DELETING OLD GRAPHS' FOLDER AND MAKING A NEW
%ONE AND CREATING A BLANK REPORT FOR EXECUTION

files = dir(fullfile(d, '*.csv'));
f=strcat(d,'\',{files.name});
n_tr=length(f);
pass_vect=strings(n_tr,1);
fail_vect=zeros(n_tr,1);
fail_count=zeros(n_tr,1);
ev_points=zeros(n_tr,1);
mark_red=zeros(n_tr,1);
std_dev_tr=zeros(n_tr,1);
std_dev_low=zeros(n_tr,1);
mark_ex=zeros(n_tr,1);
fullname=strings(n_tr,1);
testname=strings(n_tr,1);
[n_valid_tr, ~]=size(test_tr);

%creates the addresses and copying the report file
plotsdir_ex=strcat(graphs_dir,'\plots_ex');
execution_templ=strcat(template_dir, "\reportexecution.xlsx");
copyfile(execution_templ, reports_dir)
execution_file=strcat(reports_dir, "\reportexecution.xlsx");

%gets the name of the directories, deletes existing and makes a new
%plots' directory for execution
dirname_ex = dir(d);
dfolders_ex = dirname_ex([dirname_ex(:).isdir]);
dfolders_ex = dfolders_ex(~ismember({dfolders_ex(:).name},{'.','..'}));
sfold_ex=size(dfolders_ex);
names=strings(sfold_ex);
for h=1:sfold_ex
    names(h)=strcat(dfolders_ex(h).folder,"\",dfolders_ex(h).name);
end
if contains(names,"plots_ex")
    rmdir(plotsdir_ex,'s');
end
mkdir(plotsdir_ex);
perc_count=0;
perc_ex=0;

%IMPORTING NECESSARY DATA SETS FROM FILES
%for loop runs through the track files in the execution folder
for i=1:n_tr
```

```
        b_prov=char(convertStringsToChars(f(i)));
        execution_prov=readtable(b_prov);   %imports the complete .csv
        chan_title=string(execution_prov.Properties.VariableNames(:));  %saves the
variable names
        execution=readtable(b_prov,'NumHeaderLines',8); %imports the data without
unecessary informations
        execution=execution{:,:};
        [~, clm_ex]=size(execution);

        %for loop that splits the table into arrays for each variable
        for ii=1:clm_ex
            n=execution(:,1);
            t=execution(:,2);
            if contains(chan_title(ii), ["speed", "Speed"])
                v=execution(:,ii);
            elseif contains(chan_title(ii), ["latitude", "Latitude"])
                lt=execution(:,ii);
            elseif contains(chan_title(ii), ["longitude", "Longitude"])
                lg=execution(:,ii);
            elseif contains(chan_title(ii), ["brake", "Brake"])
                brake=execution(:,ii);
            end
        end

        [n_points, ~]=size(n);  %counts the number of points on the time serie
        fail_points=zeros(n_points(1),1);
        v_up=zeros(n_points(1),1);
        v_down=zeros(n_points(1),1);
        v_ideal=zeros(n_points(1),1);
        std_dev=zeros(n_points(1),1);
        std_low=zeros(n_points(1),1);

        %gets the name of the file and cuts it to name the graphs
        fullname(i)=string(files(i).name);
        testname(i)=erase(fullname(i),'.csv');
        testname(i)=strrep(testname(i),'_',' ');
        b=b_prov(end-20:end);   %takes the last 20 letters of the file name where the
track name is

        %IF  AND ELSEIF STATEMENT TO CHECK WHICH IF THE LOADED TRACK AND
        %CALCULATING THE EXECUTION PARAMETERS ACCORDING TO EACH TRACK
        %TOGETHER WITH CREATION AND SAVE OF THE CORRESPONDING GRAPH

        %for langhe track is given a 101 code just to know that has to be
        %skipped
        if sum(and(contains(test_tr,"LNG"),contains(b,"LNG")))==1
            fail_vect(i)=101;


        elseif sum(and(contains(test_tr,"PABA"),contains(b,"PABA")))==1

            nlt=zeros(5,1);
            nlg=zeros(5,1);
            %the points in the time serie with the closest longitude and
            %latitude values compared to fixed coordinates are taken
            [~, nlt(1)]=min(abs(lt-lt_paba(1)));
            [~, nlt(2)]=min(abs(lt-lt_paba(2)));
            [~, nlt(3)]=min(abs(lt-lt_paba(3)));
            [~, nlt(4)]=min(abs(lt-lt_paba(4)));
```

```
            [~, nlt(5)]=min(abs(lt-lt_paba(5)));
            [~, nlg(1)]=min(abs(lg-lg_paba(1)));
            [~, nlg(2)]=min(abs(lg-lg_paba(2)));
            [~, nlg(3)]=min(abs(lg-lg_paba(3)));
            [~, nlg(4)]=min(abs(lg-lg_paba(4)));
            [~, nlg(5)]=min(abs(lg-lg_paba(5)));

            %sets the ideal and upper and lower limit for speed according
            %to standards
            v_ideal(nlt(1):nlt(2))=27.5;
            v_ideal(nlt(4):nlt(5))=22.5;
            v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+2.5)*1.05;
            v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-2.5)*0.95;
            v_up(nlt(4):nlt(5))=(v_ideal(nlt(4):nlt(5))+2.5)*1.05;
            v_down(nlt(4):nlt(5))=(v_ideal(nlt(4):nlt(5))-2.5)*0.95;

            %for loop runs through the points of the test
            for j=nlt(1):nlt(2)
                if or(v(j)<v_down(j),v(j)>v_up(j))  %checks that the speed is in the
limits
                    fail_vect(i)=1; %sets that the test is failed
                    fail_points(j)=t(j);    %saves the failing points
                    fail_count(i)=fail_count(i)+1;  %counts the failing points
                end
            end

            for j=nlt(4):nlt(5)
                if or(v(j)<v_down(j),v(j)>v_up(j))
                    fail_vect(i)=1;
                    fail_points(j)=t(j);
                    fail_count(i)=fail_count(i)+1;
                end
            end

            %checks that there are not failing points for speed
            if fail_vect(i)~=1
                %checks if the time limit is respected
                if or(t(nlt(5))-t(nlt(1))<58*0.9, t(nlt(5))-t(nlt(1))>58*1.1)
                    fail_vect(i)=2;
                end
            end

            %creates graph and saves it
            graph_t=strcat(testname(i));
            fig = figure('visible','off');
            plot(t,v,'--','color','b')
            hold on
            plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
            hold on
            plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
            hold on
            plot(t(nlt(4):nlt(5)),v_up(nlt(4):nlt(5)),'--','color','g')
            hold on
            plot(t(nlt(4):nlt(5)),v_down(nlt(4):nlt(5)),'--','color','g')
            hold on
            for k=1:size(n)
                if fail_points(k)~=0
                    plot(fail_points(k),v(k),'x','color','r')
                end
```

```
            end
            title(graph_t);
            ylabel('Speed [km/h]');
            xlabel('Time [s]');
            legend('Speed','Upper limit', 'Lower limit','location','best')
            hold off
            figname=strcat(graph_t, '.fig');
            savefig(fig, fullfile(plotsdir_ex, figname));
            close (fig)

            %if no fail or failing for speed limit
            if or(fail_vect(i)==0,fail_vect(i)==1)
                %for loops calculate the standard deviation for each point
                %from the ideal speed and also saves the standard deviation
                %of the upper limit as the worst one in order to have a
                %reference for giving a mark
                for j=nlt(1):nlt(2)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlt(4):nlt(5)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end

                %calculates the mean among all the tested points of the
                %standard deviation
                ev_points(i)=nlt(5)-nlt(4)+nlt(2)-nlt(1)+2;
                testpoints=zeros(ev_points(i),1);
                testpoints(1:(nlt(2)-nlt(1))+1)=std_dev(nlt(1):nlt(2));
                testpoints((nlt(2)-nlt(1))+2:end)=std_dev(nlt(4):nlt(5));

                testpoints_low=zeros(nlt(5)-nlt(4)+nlt(2)-nlt(1)+2,1);
                testpoints_low(1:(nlt(2)-nlt(1))+1)=std_low(nlt(1):nlt(2));
                testpoints_low((nlt(2)-nlt(1))+2:end)=std_low(nlt(4):nlt(5));
                std_dev_tr(i)=mean(testpoints)^0.5;
                std_dev_low(i)=mean(testpoints_low)^0.5;
            end

        elseif sum(and(contains(test_tr,"APES"),contains(b,"APES")))==1

            nlt=zeros(2,1);
            nlg=zeros(2,1);
            %the points in the time serie with the closest longitude and
            %latitude values compared to fixed coordinates are taken
            [~, nlt(1)]=min(abs(lt-lt_apes(1)));
            [~, nlt(2)]=min(abs(lt-lt_apes(2)));
            [~, nlg(1)]=min(abs(lg-lg_apes(1)));
            [~, nlg(2)]=min(abs(lg-lg_apes(2)));
```

```
%sets the ideal and upper and lower limit for speed according
%to standards
v_ideal(nlt(1):nlt(2))=17.5;
v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+2.5)*1.05;
v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-2.5)*0.95;

%for loop runs through the points of the test
%checks that the speed is in the limits
%sets that the test is failed
%saves the failing points
%counts the failing points
for j=nlt(1):nlt(2)
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

%checks that there are not failing points for speed
if fail_vect(i)~=1
    %checks if the time limit is respected
    if or(t(nlt(2))-t(nlt(1))<93*0.95, t(nlt(2))-t(nlt(1))>93*1.05)
        fail_vect(i)=2;
    end
end

%creates graph and saves it
graph_t=strcat(testname(i));
fig = figure('visible','off');
plot(t,v,'--','color','b')
hold on
plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
hold on
plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
hold on
for k=1:size(n)
    if fail_points(k)~=0
        plot(fail_points(k),v(k),'x','color','r')
    end
end
title(graph_t);
ylabel('Speed [km/h]');
xlabel('Time [s]');
legend('Speed','Upper limit', 'Lower limit','location','best')
hold off
figname=strcat(graph_t, '.fig');
savefig(fig, fullfile(plotsdir_ex, figname));
close (fig)

%if no fail or failing for speed limit
if or(fail_vect(i)==0,fail_vect(i)==1)
    %for loops calculate the standard deviation for each point
    %from the ideal speed
    for j=nlt(1):nlt(2)
        if v_ideal(j)==0
            std_low(j)=0;
            std_dev(j)=0;
```

```
                else
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
            end

            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(2)-nlt(1);
            std_dev_low(i)=mean(std_low(nlt(1):nlt(2)))^0.5;
            std_dev_tr(i)=mean(std_dev(nlt(1):nlt(2)))^0.5;
        end

    elseif sum(and(contains(test_tr,"PORF-PROC"),contains(b,"PORF-PROC")))==1

        nlt=zeros(2,1);
        nlg=zeros(2,1);
        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken
        [~, nlt(1)]=min(abs(lt-lt_porfproc(1)));
        [~, nlt(2)]=min(abs(lt-lt_porfproc(2)));
        [~, nlg(1)]=min(abs(lg-lg_porfproc(1)));
        [~, nlg(2)]=min(abs(lg-lg_porfproc(2)));

        %sets the ideal and upper and lower limit for speed according
        %to standards
        v_ideal(nlt(1):nlt(2))=50;
        v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+3)*1.05;
        v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-3)*0.95;

        %for loop runs through the points of the test
        %checks that the speed is in the limits
        %sets that the test is failed
        %saves the failing points
        %counts the failing points
        for j=nlt(1):nlt(2)
            if or(v(j)<v_down(j),v(j)>v_up(j))
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        %checks that there are not failing points for speed
        if fail_vect(i)~=1
            %checks if the time limit is respected
            if or(t(nlt(2))-t(nlt(1))<29*0.93, t(nlt(2))-t(nlt(1))>29*1.07)
                fail_vect(i)=1;
            end
        end

        %creates graph and saves it
        graph_t=strcat(testname(i));
        fig = figure('visible','off');
        plot(t,v,'--','color','b')
        hold on
        plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
        hold on
        plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
```

```
            hold on
            for k=1:size(n)
                if fail_points(k)~=0
                    plot(fail_points(k),v(k),'x','color','r')
                end
            end
            title(graph_t);
            ylabel('Speed [km/h]');
            xlabel('Time [s]');
            legend('Speed','Upper limit', 'Lower limit','location','best')
            hold off
            figname=strcat(graph_t, '.fig');
            savefig(fig, fullfile(plotsdir_ex, figname));
            close (fig)


            %if no fail or failing for speed limit
            if or(fail_vect(i)==0,fail_vect(i)==1)
                %for loops calculate the standard deviation for each point
                %from the ideal speed
                for j=nlt(1):nlt(2)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end

                %calculates the mean among all the tested points of the
                %standard deviation
                ev_points(i)=nlt(2)-nlt(1);
                std_dev_low(i)=mean(std_low(nlt(1):nlt(2)))^0.5;
                std_dev_tr(i)=mean(std_dev(nlt(1):nlt(2)))^0.5;
            end

        elseif sum(and(contains(test_tr,"OB-PROC"),contains(b,"OB-PROC")))==1

            nlt=zeros(3,1);
            nlg=zeros(3,1);
            %the points in the time serie with the closest longitude and
            %latitude values compared to fixed coordinates are taken
            [~, nlt(1)]=min(abs(lt-lt_obproc(1)));
            [~, nlt(2)]=min(abs(lt-lt_obproc(2)));
            [~, nlt(3)]=min(abs(lt-lt_obproc(3)));
            [~, nlg(1)]=min(abs(lg-lg_obproc(1)));
            [~, nlg(2)]=min(abs(lg-lg_obproc(2)));
            [~, nlg(3)]=min(abs(lg-lg_obproc(3)));


            %sets the ideal and upper and lower limit for speed according
            %to standards
            v_ideal(nlt(1))=7.5;

            if or(v(nlt(1))<(v_ideal(nlt(1))-3)*0.95,v(nlt(1))>(v_ideal(nlt(1))+3)*1.05)
                for ttt=nlt(1):n_points
                    if and(v(ttt)>(v_ideal(nlt(1))-
3)*0.95,v(ttt)<(v_ideal(nlt(1))+3)*1.05)
```

```
                nlt(1)=ttt;
                nlg(1)=ttt;
                break
            end
        end
    end

    v_ideal(nlt(1))=7.5;
    v_ideal(nlt(2))=30;
    v_ideal(nlt(3))=60;
    m1_id=(v_ideal(nlt(2))-v_ideal(nlt(1)))/(nlt(2)-nlt(1));
    m2_id=(v_ideal(nlt(3))-v_ideal(nlt(2)))/(nlt(3)-nlt(2));

    for j=1:(nlt(2)-nlt(1))
        v_ideal(j+nlt(1))=m1_id*j+v_ideal(nlt(1));
    end
    for j=1:(nlt(3)-nlt(2))
        v_ideal(j+nlt(2))=m2_id*j+v_ideal(nlt(2));
    end

    v_up(nlt(1))=(v_ideal(nlt(1))+3)*1.05;
    v_down(nlt(1))=(v_ideal(nlt(1))-3)*0.95;
    v_up(nlt(2))=(v_ideal(nlt(2))+3)*1.05;
    v_down(nlt(2))=(v_ideal(nlt(2))-3)*0.95;
    v_up(nlt(3))=(v_ideal(nlt(3))+3)*1.05;
    v_down(nlt(3))=(v_ideal(nlt(3))-3)*0.95;
    v_up(nlt(1)+1:nlt(2)-1)=(v_ideal(nlt(1)+1:nlt(2)-1)+5)*1.05;
    v_down(nlt(1)+1:nlt(2)-1)=(v_ideal(nlt(1)+1:nlt(2)-1)-5)*0.95;
    v_up(nlt(2)+1:nlt(3)-1)=(v_ideal(nlt(2)+1:nlt(3)-1)+5)*1.05;
    v_down(nlt(2)+1:nlt(3)-1)=(v_ideal(nlt(2)+1:nlt(3)-1)-5)*0.95;

    %checks the entrance speed
    if or(v(nlt(1))<(v_ideal(nlt(1))-3)*0.95,v(nlt(1))>(v_ideal(nlt(1))+3)*1.05)
        fail_vect(i)=1;
        fail_points(nlt(1))=t(nlt(1));
    end

    %for loop runs through the points of the test
    %checks that the speed is in the limits
    %sets that the test is failed
    %saves the failing points
    %counts the failing points
    for j=nlt(1)+1:nlt(2)-1
        if or(v(j)<v_down(j),v(j)>v_up(j))
            fail_vect(i)=1;
            fail_points(j)=t(j);
            fail_count(i)=fail_count(i)+1;
        end
    end

    if or(v(nlt(2))<(v_ideal(nlt(2))-3)*0.95,v(nlt(2))>(v_ideal(nlt(2))+3)*1.05)
        fail_vect(i)=1;
        fail_points(nlt(2))=t(nlt(2));
        fail_count(i)=fail_count(i)+1;
    end

    for j=nlt(2)+1:nlt(3)-1
        if or(v(j)<v_down(j),v(j)>v_up(j))
            fail_vect(i)=1;
```

```
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        if or(v(nlt(3))<(v_ideal(nlt(3))-3)*0.95,v(nlt(3))>(v_ideal(nlt(3))+3)*1.05)
            fail_vect(i)=1;
            fail_points(nlt(3))=t(nlt(3));
                fail_count(i)=fail_count(i)+1;
        end

        %checks that there are not failing points for speed
        if fail_vect(i)~=1
            v1=v(nlt(1):nlt(2));
            n1=n(nlt(1):nlt(2));
            v2=v(nlt(2):nlt(3));
            n2=n(nlt(2):nlt(3));
            m1=polyfit(v1,n1,1);
            m2=polyfit(v2,n2,1);

            %checks if there are positive accelerations as the standard
            %requires
            if or(m1(1)<=0, m2(1)<=0)
                fail_vect(i)=1;
            end
        end

        %creates graph and saves it
        graph_t=strcat(testname(i));
        fig = figure('visible','off');
        plot(t,v,'--','color','b')
        hold on
        plot(t(nlt(1):nlt(3)),v_up(nlt(1):nlt(3)),'--','color','g')
        hold on
        plot(t(nlt(1):nlt(3)),v_down(nlt(1):nlt(3)),'--','color','g')
        hold on
        for k=1:size(n)
            if fail_points(k)~=0
                plot(fail_points(k),v(k),'x','color','r')
            end
        end
        title(graph_t);
        ylabel('Speed [km/h]');
        xlabel('Time [s]');
        legend('Speed','Upper limit', 'Lower limit','location','best')
        hold off
        figname=strcat(graph_t, '.fig');
        savefig(fig, fullfile(plotsdir_ex, figname));
        close (fig)

        %if no fail or failing for speed limit
        if or(fail_vect(i)==0,fail_vect(i)==1)
            %for loops calculate the standard deviation for each point
            %from the ideal speed
            for j=nlt(1):nlt(3)
                if v_ideal(j)==0
                    std_low(j)=0;
                    std_dev(j)=0;
                else
```

```
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
            end

            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(3)-nlt(1);
            std_dev_low(i)=mean(std_low(nlt(1):nlt(3)))^0.5;
            std_dev_tr(i)=mean(std_dev(nlt(1):nlt(3)))^0.5;
        end

    elseif sum(and(contains(test_tr,"PLX"),contains(b,"PLX")))==1

        nlt=zeros(10,1);
        nlg=zeros(10,1);
        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken with
        %particular care considering that the PLX track is not a
        %straight line, but a loop
        [~, n_mx1]=max(lg(1:round(n_points/2,0)));
        [~, n_mx2]=max(lg(round(n_points/2,0):end));
        n_mx2=n_mx2+round(n_points/2,0);
        [~, n_mn1]=min(lg(n_mx1:n_mx2));
        n_mn1=n_mn1+n_mx1;

        lg_prov=lg(1:n_mx1);
        [~, nlg(1)]=min(abs(lg_prov-lg_plx(1)));
        [~, nlg(2)]=min(abs(lg_prov-lg_plx(2)));
        [~, nlg(3)]=min(abs(lg_prov-lg_plx(3)));
        [~, nlg(4)]=min(abs(lg_prov-lg_plx(4)));
        [~, nlg(5)]=min(abs(lg_prov-lg_plx(5)));
        [~, nlg(6)]=min(abs(lg_prov-lg_plx(6)));

        lg_prov=lg(n_mx1:n_mn1);
        [~, nlg(7)]=min(abs(lg_prov-lg_plx(7)));
        nlg(7)=nlg(7)+n_mx1;

        lg_prov=lg(n_mn1:n_mx2);
        [~, nlg(8)]=min(abs(lg_prov-lg_plx(8)));
        nlg(8)=nlg(8)+n_mn1;

        lg_prov=lg(n_mx2:end);
        [~, nlg(9)]=min(abs(lg_prov-lg_plx(9)));
        [~, nlg(10)]=min(abs(lg_prov-lg_plx(10)));
        nlg(9)=nlg(9)+n_mx2;
        nlg(10)=nlg(10)+n_mx2;

        %sets the ideal and upper and lower limit for speed according
        %to standards
        v_ideal(nlg(1):nlg(2))=50;
        v_ideal(nlg(3):nlg(4))=30;
        v_ideal(nlg(5):nlg(6))=50;
        v_ideal(nlg(6):nlg(7))=40;
        v_ideal(nlg(7):nlg(8))=50;
        v_ideal(nlg(8):nlg(9))=40;
        v_ideal(nlg(9):nlg(10))=50;
        v_up(nlg(1):nlg(2))=(v_ideal(nlg(1):nlg(2))+3)*1.05;
        v_down(nlg(1):nlg(2))=(v_ideal(nlg(1):nlg(2))-3)*0.95;
```

```
v_up(nlg(3):nlg(4))=(v_ideal(nlg(3):nlg(4))+3)*1.05;
v_down(nlg(3):nlg(4))=(v_ideal(nlg(3):nlg(4))-3)*0.95;
v_up(nlg(5):nlg(6))=(v_ideal(nlg(5):nlg(6))+3)*1.05;
v_down(nlg(5):nlg(6))=(v_ideal(nlg(5):nlg(6))-3)*0.95;
v_up(nlg(6):nlg(7))=(v_ideal(nlg(6):nlg(7))+3)*1.05;
v_down(nlg(6):nlg(7))=(v_ideal(nlg(6):nlg(7))-3)*0.95;
v_up(nlg(7):nlg(8))=(v_ideal(nlg(7):nlg(8))+3)*1.05;
v_down(nlg(7):nlg(8))=(v_ideal(nlg(7):nlg(8))-3)*0.95;
v_up(nlg(8):nlg(9))=(v_ideal(nlg(8):nlg(9))+3)*1.05;
v_down(nlg(8):nlg(9))=(v_ideal(nlg(8):nlg(9))-3)*0.95;
v_up(nlg(9):nlg(10))=(v_ideal(nlg(9):nlg(10))+3)*1.05;
v_down(nlg(9):nlg(10))=(v_ideal(nlg(9):nlg(10))-3)*0.95;

%for loop runs through the points of the test
%checks that the speed is in the limits
%sets that the test is failed
%saves the failing points
%counts the failing points
for j=nlg(1):nlg(2)-1
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

for j=nlg(3)+1:nlg(4)-1
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

for j=nlg(5)+1:nlg(6)-1
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

for j=nlg(6)+1:nlg(7)-1
    if and(v(j)>v_down(j),v(j)<v_up(j))
        break
    elseif and(or(v(j)<v_down(j),v(j)>v_up(j)),j==nlg(7))
        fail_vect(i)=1;
        fail_points(j)=t(nlg(6)+round((nlg(7)-nlg(6))/2,0));
        fail_count(i)=fail_count(i)+1;
    end
end

for j=nlg(7)+1:nlg(8)-1
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end
```

```
for j=nlg(8)+1:nlg(9)-1
    if and(v(j)>v_down(j),v(j)<v_up(j))
        break
    elseif and(or(v(j)<v_down(j),v(j)>v_up(j)),j==nlg(9))
        fail_vect(i)=1;
        fail_points(j)=t(nlg(8)+round((nlg(9)-nlg(8))/2,0));
        fail_count(i)=fail_count(i)+1;
    end
end
for j=nlg(9)+1:nlg(10)
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

%creates graph and saves it
graph_t=strcat(testname(i));
fig = figure('visible','off');
plot(t,v,'--','color','b')
hold on
plot(t(nlg(1):nlg(2)),v_up(nlg(1):nlg(2)),'--','color','g')
hold on
plot(t(nlg(1):nlg(2)),v_down(nlg(1):nlg(2)),'--','color','g')
hold on
plot(t(nlg(3):nlg(4)),v_up(nlg(3):nlg(4)),'--','color','g')
hold on
plot(t(nlg(3):nlg(4)),v_down(nlg(3):nlg(4)),'--','color','g')
hold on
plot(t(nlg(5):nlg(10)),v_up(nlg(5):nlg(10)),'--','color','g')
hold on
plot(t(nlg(5):nlg(10)),v_down(nlg(5):nlg(10)),'--','color','g')
hold on
for k=1:size(n)
    if fail_points(k)~=0
        plot(fail_points(k),v(k),'x','color','r')
    end
end
title(graph_t);
ylabel('Speed [km/h]');
xlabel('Time [s]');
legend('Speed','Upper limit', 'Lower limit','location','best')
hold off
figname=strcat(graph_t, '.fig');
savefig(fig, fullfile(plotsdir_ex, figname));
close (fig)

%if no fail or failing for speed limit
if or(fail_vect(i)==0,fail_vect(i)==1)
    %for loops calculate the standard deviation for each point
    %from the ideal speed
    for j=nlg(1):nlg(2)
        if v_ideal(j)==0
            std_low(j)=0;
            std_dev(j)=0;
        else
            std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
```

```
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(3):nlg(4)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(5):nlg(6)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(7):nlg(8)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(9):nlg(10)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end

                %calculates the mean among all the tested points of the
                %standard deviation
                ev_points(i)=nlg(10)-nlg(9)+nlg(8)-nlg(7)+nlg(6)-nlg(5)+nlg(4)-
nlg(3)+nlg(2)-nlg(1)+5;
                testpoints=zeros(ev_points(i),1);
                testpoints(1:(nlg(2)-nlg(1))+1)=std_dev(nlg(1):nlg(2));
                testpoints((nlg(2)-nlg(1))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3))+2)=std_dev(nlg(3):nlg(4));
                testpoints((nlg(2)-nlg(1)+nlg(4)-nlg(3))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3)+nlg(6)-nlg(5))+2)=std_dev(nlg(5):nlg(6));
                testpoints((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5))+2:(nlg(2)-
nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7))+2)=std_dev(nlg(7):nlg(8));
                testpoints((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-
nlg(7))+2:(nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7)+nlg(10)-
nlg(9))+2)=std_dev(nlg(9):nlg(10));

                testpoints_low=zeros(nlg(10)-nlg(9)+nlg(8)-nlg(7)+nlg(6)-nlg(5)+nlg(4)-
nlg(3)+nlg(2)-nlg(1)+5,1);
                testpoints_low(1:(nlg(2)-nlg(1))+1)=std_low(nlg(1):nlg(2));
```

```
                testpoints_low((nlg(2)-nlg(1))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3))+2)=std_low(nlg(3):nlg(4));
                testpoints_low((nlg(2)-nlg(1)+nlg(4)-nlg(3))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3)+nlg(6)-nlg(5))+2)=std_low(nlg(5):nlg(6));
                testpoints_low((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5))+2:(nlg(2)-
nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7))+2)=std_low(nlg(7):nlg(8));
                testpoints_low((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-
nlg(7))+2:(nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7)+nlg(10)-
nlg(9))+2)=std_low(nlg(9):nlg(10));
                std_dev_tr(i)=mean(testpoints)^0.5;
                std_dev_low(i)=mean(testpoints_low)^0.5;
            end

        elseif sum(and(contains(test_tr,"MCISA"),contains(b,"MCISA")))==1

            nlt=zeros(2,1);
            nlg=zeros(2,1);
            %for to find the starting point of the acceleration of mcisa
            check1=0;
            for ttt=1:n_points
                if (v(ttt)>0.95*80&&lt(ttt)>(lt_mcisa(2)-
0.0001)&&lt(ttt)<(lt_mcisa(2)+0.0001)&&check1==0)
                    check1=1;
                    ttt1=ttt;
                elseif (v(ttt)>0.99*80&&lt(ttt)>(lt_mcisa(2)-
0.0001)&&lt(ttt)<(lt_mcisa(2)+0.0001)&&check1==1)
                    check1=2;
                    lt=lt(1:ttt);
                    lg=lg(1:ttt);
                    break
                end
                if (check1==0&&lt(ttt)>(lt_mcisa(2)+0.0001))
                    break
                end
            end

            if check1==0
                fail_vect(i)=1;
            elseif check1==1
                lt=lt(1:ttt1);
                lg=lg(1:ttt1);
            end
            for j=1:n_points
                if v(j)>1
                    ttt2=j;
                    break
                end
            end

            lt=lt(ttt2:ttt1);
            lt_prov=lt;
            lg=lg(ttt2:ttt1);
            check_t1=0;
            while check_t1==0
                [~, nlt(1)]=min(abs(lt_prov-lt_mcisa(1)));
                if v(nlt(1))<3
                    check_t1=1;
                else
                    lt_prov(nlt(1))=0;
```

```
    end
end

%the points in the time serie with the closest longitude and
%latitude values compared to fixed coordinates are taken
[~, nlt(2)]=min(abs(lt-lt_mcisa(2)));
nlt(1)=nlt(1)+ttt2;
nlt(2)=nlt(2)+ttt2;

%sets the ideal and upper and lower limit for speed according
%to standards
v_ideal(nlt(1))=0;
v_ideal(nlt(2))=80;
m1_id=(v_ideal(nlt(2))-v_ideal(nlt(1)))/(nlt(2)-nlt(1));

for j=1:(nlt(2)-nlt(1))
    v_ideal(j+nlt(1))=m1_id*j+v_ideal(nlt(1));
end

v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+7.5)*1.05;
v_down(nlt(1):nlt(2))=v_ideal(nlt(1));

%checks entering point's speed
if v(nlt(1))>v_up(nlt(1))
    fail_vect(i)=1;
    fail_points(nlt(1))=t(nlt(1));
    fail_count(i)=fail_count(i)+1;
end

%for loop runs through the points of the test
%checks that the speed is in the limits
%sets that the test is failed
%saves the failing points
%counts the failing points
for j=nlt(1)+1:nlt(2)-1
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

if or(v(nlt(2))<(v_ideal(nlt(2))-3)*0.95,v(nlt(2))>(v_ideal(nlt(2))+3)*1.05)
    fail_vect(i)=1;
    fail_points(nlt(2))=t(nlt(2));
    fail_count(i)=fail_count(i)+1;
end

%checks that there are not failing points for speed
if fail_vect(i)~=1
    v1=v(nlt(1):nlt(2));
    n1=n(nlt(1):nlt(2));
    m1=polyfit(v1,n1,1);
    %checks the positive acceleration
    if m1(1)<=0
        fail_vect(i)=1;
    end
end
```

```
            %creates graph and saves it
            graph_t=strcat(testname(i));
            fig = figure('visible','off');
            plot(t(nlt(1):nlt(2)),v(nlt(1):nlt(2)),'--','color','b')
            hold on
            plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
            hold on
            plot(t(nlt(1):nlt(2)),v_ideal(nlt(1):nlt(2)),'-.','color','r')
            hold on
            for k=1:size(n)
                if fail_points(k)~=0
                    plot(fail_points(k),v(k),'x','color','r')
                end
            end
            title(graph_t);
            ylabel('Speed [km/h]');
            xlabel('Time [s]');
            legend('Speed','Upper limit', 'Ideal speed','location','best')
            hold off
            figname=strcat(graph_t, '.fig');
            savefig(fig, fullfile(plotsdir_ex, figname));
            close (fig)

            v=v+30;
            v_up=v_up+30;
            v_ideal=v_ideal+30;

            %if no fail or failing for speed limit
            if or(fail_vect(i)==0,fail_vect(i)==1)
                %for loops calculate the standard deviation for each point
                %from the ideal speed
                for j=nlt(1):nlt(2)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                %calculates the mean among all the tested points of the
                %standard deviation
                ev_points(i)=nlt(2)-nlt(1);
                std_dev_low(i)=mean(std_low(nlt(1):nlt(2)))^0.5;
                std_dev_tr(i)=mean(std_dev(nlt(1):nlt(2)))^0.5;
            end

        elseif sum(and(contains(test_tr,"AV"),contains(b,"AV")))==1
            max_br=zeros(3,1);
            start_brake=zeros(3,1);
            start_dec=zeros(3,1);
            end_dec=zeros(3,1);
            valid_brakes=0;

            %for loop finds the first point over 80km/h
            for j=n(1):n(end)
                if v(j)>0.95*80
                    n1_av=j;
                    break
```

```
                end
            end
            %if a point over 80km/h was found
            if n1_av==j
                j=n1_av;
                %while loop looks for a 0km/h velocity point after the
                %80km/h point
                while j<n(end)
                    if v(j)==0
                        end_dec(valid_brakes+1)=j;
                        for jj=end_dec(valid_brakes+1):-1:n1_av
                            if and(v(jj)>0.95*80, ismember(jj,start_brake)==0)
                                start_dec(valid_brakes+1)=jj;
                                break
                            elseif and(v(jj)>0.95*80, ismember(jj,start_brake)==1)
                                break
                            end
                        end
                        if ismember(jj,start_brake)==0
                            start_brake(valid_brakes+1)=jj;
                            check_abs=0;
                            for jj=start_dec(valid_brakes+1):end_dec(valid_brakes+1)
                                %checks that there is a breaking pressure
                                %higher than 75%
                                if brake(jj)>75
                                    check_abs=check_abs+1;
                                    if check_abs==50

max_br(valid_brakes+1)=max(brake(start_dec(valid_brakes+1):end_dec(valid_brakes+1)));
                                        valid_brakes=valid_brakes+1;
                                        break
                                    end
                                end
                            end
                        end
                        %checks that the former loop finds 3 valid brakings
                        %from 80km/h to 0km/h as the standard requires
                        if valid_brakes==3
                            break
                        end
                        for jjj=j:n(end)
                            if v(jjj)~=0
                                j=jjj;
                                break
                            end
                        end
                    end
                    j=j+1;
                end
            end

            %creates graph and saves it
            graph_t=strcat(testname(i));
            fig = figure('visible','off');
            plot(t,v,'--','color','b')
            hold on
            for av_i=1:valid_brakes

plot(t(start_dec(av_i):end_dec(av_i)),v_up(start_dec(av_i):end_dec(av_i)),'--
```

```
','color','g')
                hold on
            end
            title(graph_t);
            ylabel('Speed [km/h]');
            xlabel('Time [s]');
            legend('Speed','Braking sections','location','best')
            hold off
            figname=strcat(graph_t, '.fig');
            savefig(fig, fullfile(plotsdir_ex, figname));
            close (fig)

            %if less than 3 valid brakes, the test fails
            if valid_brakes<3
                fail_vect(i)=1;
            end

        elseif sum(and(contains(test_tr,"OLC-LAS"),contains(b,"OLC-LAS")))==1

            nlt=zeros(4,1);
            nlg=zeros(4,1);
            %the points in the time serie with the closest longitude and
            %latitude values compared to fixed coordinates are taken
            [~, mn_olclas]=min(lt);
            lt_prov=lt(1:mn_olclas);
            lg_prov=lg(1:mn_olclas);

            [~, nlt(1)]=min(abs(lt_prov-lt_olclas(1)));
            [~, nlt(2)]=min(abs(lt_prov-lt_olclas(2)));
            [~, nlg(1)]=min(abs(lg_prov-lg_olclas(1)));
            [~, nlg(2)]=min(abs(lg_prov-lg_olclas(2)));

            lt_prov=lt(mn_olclas:end);
            lg_prov=lg(mn_olclas:end);
            [~, nlt(3)]=min(abs(lt_prov-lt_olclas(3)));
            [~, nlt(4)]=min(abs(lt_prov-lt_olclas(4)));
            [~, nlg(3)]=min(abs(lg_prov-lg_olclas(3)));
            [~, nlg(4)]=min(abs(lg_prov-lg_olclas(4)));
            nlt(3)=nlt(3)+mn_olclas;
            nlt(4)=nlt(4)+mn_olclas;
            nlg(3)=nlg(3)+mn_olclas;
            nlg(4)=nlg(4)+mn_olclas;

            %sets the ideal and upper and lower limit for speed according
            %to standards
            v_ideal(nlt(1):nlt(2))=50;
            v_ideal(nlt(3):nlt(4))=30;
            v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+3)*1.05;
            v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-3)*0.95;
            v_up(nlt(3):nlt(4))=(v_ideal(nlt(3):nlt(4))+3)*1.05;
            v_down(nlt(3):nlt(4))=(v_ideal(nlt(3):nlt(4))-3)*0.95;

            %for loop runs through the points of the test
            %checks that the speed is in the limits
            %sets that the test is failed
            %saves the failing points
            %counts the failing points
            for j=nlt(1):nlt(2)
                if or(v(j)<v_down(j),v(j)>v_up(j))
```

```
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end
        for j=nlt(3):nlt(4)
            if or(v(j)<v_down(j),v(j)>v_up(j))
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        %creates graph and saves it
        graph_t=strcat(testname(i));
        fig = figure('visible','off');
        plot(t,v,'--','color','b')
        hold on
        plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
        hold on
        plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
        hold on
        plot(t(nlt(3):nlt(4)),v_up(nlt(3):nlt(4)),'--','color','g')
        hold on
        plot(t(nlt(3):nlt(4)),v_down(nlt(3):nlt(4)),'--','color','g')
        hold on
        for k=1:size(n)
            if fail_points(k)~=0
                plot(fail_points(k),v(k),'x','color','r')
            end
        end
        title(graph_t);
        ylabel('Speed [km/h]');
        xlabel('Time [s]');
        legend('Speed','Upper limit', 'Lower limit','location','best')
        hold off
        figname=strcat(graph_t, '.fig');
        savefig(fig, fullfile(plotsdir_ex, figname));
        close (fig)

        %if no fail or failing for speed limit
        if or(fail_vect(i)==0,fail_vect(i)==1)
            for j=nlt(1):nlt(2)
                if v_ideal(j)==0
                    std_low(j)=0;
                    std_dev(j)=0;
                else
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
            end
            for j=nlt(3):nlt(4)
                if v_ideal(j)==0
                    std_low(j)=0;
                    std_dev(j)=0;
                else
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
```

```
            end
            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(4)-nlt(3)+nlt(2)-nlt(1)+2;
            testpoints=zeros(ev_points(i),1);
            testpoints(1:(nlt(2)-nlt(1))+1)=std_dev(nlt(1):nlt(2));
            testpoints((nlt(2)-nlt(1))+2:end)=std_dev(nlt(3):nlt(4));
            testpoints_low=zeros(nlt(4)-nlt(3)+nlt(2)-nlt(1)+2,1);
            testpoints_low(1:(nlt(2)-nlt(1))+1)=std_low(nlt(1):nlt(2));
            testpoints_low((nlt(2)-nlt(1))+2:end)=std_low(nlt(3):nlt(4));
            std_dev_tr(i)=mean(testpoints)^0.5;
            std_dev_low(i)=mean(testpoints_low)^0.5;
        end

    elseif sum(and(contains(test_tr,"OL-DS"),contains(b,"OL-DS")))==1

        nlt=zeros(4,1);
        nlg=zeros(4,1);
        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken
        [~, mn_olds]=min(lt);
        lt_prov=lt(1:mn_olds);
        [~, mx1_olds]=max(lt_prov);
        lt_prov=lt(mx1_olds:mn_olds);
        [~, nlt(1)]=min(abs(lt_prov-lt_olds(1)));
        [~, nlt(2)]=min(abs(lt_prov-lt_olds(2)));
        nlt(1)=nlt(1)+mx1_olds;
        nlt(2)=nlt(2)+mx1_olds;

        lt_prov=lt(mn_olds:end);
        [~, mx2_olds]=max(lt_prov);
        mx2_olds=mx2_olds+mn_olds;
        lt_prov=lt(mn_olds:mx2_olds);
        [~, nlt(3)]=min(abs(lt_prov-lt_olds(3)));
        [~, nlt(4)]=min(abs(lt_prov-lt_olds(4)));
        nlt(3)=nlt(3)+mn_olds;
        nlt(4)=nlt(4)+mn_olds;

        %sets the ideal and upper and lower limit for speed according
        %to standards
        v_ideal(nlt(1):nlt(2))=50;
        v_ideal(nlt(3):nlt(4))=50;
        v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+3)*1.05;
        v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-3)*0.95;
        v_up(nlt(3):nlt(4))=(v_ideal(nlt(3):nlt(4))+3)*1.05;
        v_down(nlt(3):nlt(4))=(v_ideal(nlt(3):nlt(4))-3)*0.95;

        %for loop runs through the points of the test
        %checks that the speed is in the limits
        %sets that the test is failed
        %saves the failing points
        %counts the failing points
        for j=nlt(1):nlt(2)
            if or(v(j)<v_down(j),v(j)>v_up(j))
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end
```

```
for j=nlt(3):nlt(4)
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end

%creates graph and saves it
graph_t=strcat(testname(i));
fig = figure('visible','off');
plot(t,v,'--','color','b')
hold on
plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
hold on
plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
hold on
plot(t(nlt(3):nlt(4)),v_up(nlt(3):nlt(4)),'--','color','g')
hold on
plot(t(nlt(3):nlt(4)),v_down(nlt(3):nlt(4)),'--','color','g')
hold on
for k=1:size(n)
    if fail_points(k)~=0
        plot(fail_points(k),v(k),'x','color','r')
    end
end
title(graph_t);
ylabel('Speed [km/h]');
xlabel('Time [s]');
legend('Speed','Upper limit', 'Lower limit','location','best')
hold off
figname=strcat(graph_t, '.fig');
savefig(fig, fullfile(plotsdir_ex, figname));
close (fig)

%if no fail or failing for speed limit
if or(fail_vect(i)==0,fail_vect(i)==1)
    %for loops calculate the standard deviation for each point
    %from the ideal speed
    for j=nlt(1):nlt(2)
        if v_ideal(j)==0
            std_low(j)=0;
            std_dev(j)=0;
        else
            std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
            std_dev(j)=(v(j)/v_ideal(j)-1)^2;
        end
    end
    for j=nlt(3):nlt(4)
        if v_ideal(j)==0
            std_low(j)=0;
            std_dev(j)=0;
        else
            std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
            std_dev(j)=(v(j)/v_ideal(j)-1)^2;
        end
    end
```

```
            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(4)-nlt(3)+nlt(2)-nlt(1)+2;
            testpoints=zeros(ev_points(i),1);
            testpoints(1:(nlt(2)-nlt(1))+1)=std_dev(nlt(1):nlt(2));
            testpoints((nlt(2)-nlt(1))+2:end)=std_dev(nlt(3):nlt(4));

            testpoints_low=zeros(nlt(4)-nlt(3)+nlt(2)-nlt(1)+2,1);
            testpoints_low(1:(nlt(2)-nlt(1))+1)=std_low(nlt(1):nlt(2));
            testpoints_low((nlt(2)-nlt(1))+2:end)=std_low(nlt(3):nlt(4));
            std_dev_tr(i)=mean(testpoints)^0.5;
            std_dev_low(i)=mean(testpoints_low)^0.5;
        end

    elseif sum(and(contains(test_tr,"TR-BU"),contains(b,"TR-BU")))==1

        nlt=zeros(3,1);
        nlg=zeros(3,1);
        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken
        [lt(1), nlt(1)]=min(abs(lt-lt_trbu(1)));
        [lt(2), nlt(2)]=min(abs(lt-lt_trbu(2)));
        [lt(3), nlt(3)]=min(abs(lt-lt_trbu(3)));
        [lg(1), nlg(1)]=min(abs(lg-lg_trbu(1)));
        [lg(2), nlg(2)]=min(abs(lg-lg_trbu(2)));
        [lg(3), nlg(3)]=min(abs(lg-lg_trbu(3)));

        %sets the ideal and upper and lower limit for speed according
        %to standards
        v_ideal(nlt(1))=5;
        v_ideal(nlt(2):nlt(3))=30;
        m1_id=(v_ideal(nlt(2))-v_ideal(nlt(1)))/(nlt(2)-nlt(1));

        for j=1:(nlt(2)-nlt(1))
            v_ideal(j+nlt(1))=m1_id*j+v_ideal(nlt(1));
        end

        v_up(nlt(2))=(v_ideal(nlt(2))+3)*1.05;
        v_down(nlt(2))=(v_ideal(nlt(2))-3)*0.95;
        v_up(nlt(3))=(v_ideal(nlt(3))+3)*1.05;
        v_down(nlt(3))=(v_ideal(nlt(3))-3)*0.95;
        v_up(nlt(1):nlt(2)-1)=(v_ideal(nlt(1):nlt(2)-1)+5)*1.05;
        v_down(nlt(1):nlt(2)-1)=(v_ideal(nlt(1):nlt(2)-1)-5)*0.95;
        v_up(nlt(2)+1:nlt(3)-1)=(v_ideal(nlt(2)+1:nlt(3)-1)+3)*1.05;
        v_down(nlt(2)+1:nlt(3)-1)=(v_ideal(nlt(2)+1:nlt(3)-1)-3)*0.95;

        %checks the entrance speed
        if or(v(nlt(1))<(v_ideal(nlt(1))-3)*0.95,v(nlt(1))>(v_ideal(nlt(1))+3)*1.05)
            fail_vect(i)=1;
            fail_points(nlt(1))=t(nlt(1));
            fail_count(i)=fail_count(i)+1;
        end
        %for loop runs through the points of the test
        %checks that the speed is in the limits
        %sets that the test is failed
        %saves the failing points
        %counts the failing points
        for j=nlt(1)+1:nlt(2)-1
            if or(v(j)<v_down(j),v(j)>v_up(j))
```

```
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        if or(v(nlt(2))<(v_ideal(nlt(2))-3)*0.95,v(nlt(2))>(v_ideal(nlt(2))+3)*1.05)
            fail_vect(i)=1;
            fail_points(nlt(2))=t(nlt(2));
            fail_count(i)=fail_count(i)+1;
        end

        for j=nlt(2)+1:nlt(3)-1
            if or(v(j)<v_down(j),v(j)>v_up(j))
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        %checks exit velocity
        if or(v(nlt(3))<(v_ideal(nlt(3))-3)*0.95,v(nlt(3))>(v_ideal(nlt(3))+3)*1.05)
            fail_vect(i)=1;
            fail_points(nlt(3))=t(nlt(3));
            fail_count(i)=fail_count(i)+1;
        end

        %checks that there is a linear acceleration
        if fail_vect(i)~=1
            v1=v(nlt(1):nlt(2));
            n1=n(nlt(1):nlt(2));
            m1=polyfit(v1,n1,1);
            if m1(1)<=0
                fail_vect(i)=1;
            end
        end

        %creates graph and saves it
        graph_t=strcat(testname(i));
        fig = figure('visible','off');
        plot(t,v,'--','color','b')
        hold on
        plot(t(nlt(1):nlt(3)),v_up(nlt(1):nlt(3)),'--','color','g')
        hold on
        plot(t(nlt(1):nlt(3)),v_down(nlt(1):nlt(3)),'--','color','g')
        hold on
        for k=1:size(n)
            if fail_points(k)~=0
                plot(fail_points(k),v(k),'x','color','r')
            end
        end
        title(graph_t);
        ylabel('Speed [km/h]');
        xlabel('Time [s]');
        legend('Speed','Upper limit', 'Lower limit','location','best')
        hold off
        figname=strcat(graph_t, '.fig');
        savefig(fig, fullfile(plotsdir_ex, figname));
        close (fig)
```

```
        %if no fail or failing for speed limit
        if or(fail_vect(i)==0,fail_vect(i)==1)
            %for loops calculate the standard deviation for each point
            %from the ideal speed
            for j=nlt(1):nlt(3)
                if v_ideal(j)==0
                    std_low(j)=0;
                    std_dev(j)=0;
                else
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
            end

            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(3)-nlt(1);
            std_dev_low(i)=mean(std_low(nlt(1):nlt(3)))^0.5;
            std_dev_tr(i)=mean(std_dev(nlt(1):nlt(3)))^0.5;
        end

    elseif sum(and(contains(test_tr,"RAL"),contains(b,"RAL")))==1

        nlt=zeros(8,1);
        nlg=zeros(8,1);
        %a reference point is set and an array saves the number of
        %times the vehicle crosses this point to split the time series
        ref_lg_low=8.298825;
        ref_lg_up=8.298835;
        ref_ral=zeros(100,1);
        ref=1;
        check_ref=0;
        for ttt=1:n_points
            if (lg(ttt)>ref_lg_low&&lg(ttt)<ref_lg_up&&check_ref==0)
                ref_ral(ref)=ttt;
                ref=ref+1;
                check_ref=1;
            end
            if ((lg(ttt)<ref_lg_low||lg(ttt)>ref_lg_up)&&check_ref==1)
                check_ref=0;
            end
        end
        ref_ral=ref_ral(1:ref-1);

        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken
        lg_prov=lg(1:ref_ral(1));
        [~, nlg(1)]=min(abs(lg_prov-lg_ral(1)));

        lg_prov=lg(ref_ral(1):ref_ral(2));
        [~, n_mx_ral1]=max(lg_prov);
        n_mx_ral1=n_mx_ral1+ref_ral(1);
        lg_prov=lg(ref_ral(1):n_mx_ral1);
        [~, nlg(2)]=min(abs(lg_prov-lg_ral(2)));
        nlg(2)=nlg(2)+ref_ral(1);
        lg_prov=lg(n_mx_ral1:ref_ral(2));
        [~, nlg(3)]=min(abs(lg_prov-lg_ral(2)));
        nlg(3)=nlg(3)+n_mx_ral1;
```

```
lg_prov=lg(ref_ral(2):ref_ral(3));
[~, n_mn_ral1]=min(lg_prov);
n_mn_ral1=n_mn_ral1+ref_ral(2);
lg_prov=lg(ref_ral(2):n_mn_ral1);
[~, nlg(4)]=min(abs(lg_prov-lg_ral(1)));
nlg(4)=nlg(4)+ref_ral(2);

lg_prov=lg(ref_ral(end-2):ref_ral(end-1));
[~, n_mn_ral2]=min(lg_prov);
n_mn_ral2=n_mn_ral2+ref_ral(end-2);
lg_prov=lg(n_mn_ral2:ref_ral(end-1));
[~, nlg(5)]=min(abs(lg_prov-lg_ral(3)));
nlg(5)=nlg(5)+n_mn_ral2;

lg_prov=lg(ref_ral(end-1):ref_ral(end));
[~, n_mx_ral2]=max(lg_prov);
n_mx_ral2=n_mx_ral2+ref_ral(end-1);
lg_prov=lg(ref_ral(end-1):n_mx_ral2);
[~, nlg(6)]=min(abs(lg_prov-lg_ral(4)));
nlg(6)=nlg(6)+ref_ral(end-1);
lg_prov=lg(n_mx_ral2:ref_ral(end));
[~, nlg(7)]=min(abs(lg_prov-lg_ral(4)));
nlg(7)=nlg(7)+n_mx_ral2;

lg_prov=lg(ref_ral(end):end);
[~, nlg(8)]=min(abs(lg_prov-lg_ral(3)));
nlg(8)=nlg(8)+ref_ral(end);

%sets the ideal and upper and lower limit for speed according
%to standards
v_ideal(nlg(1):nlg(2))=7.5;
v_up(nlg(1):nlg(2))=(v_ideal(nlg(1):nlg(2))+2.5)*1.1;
v_down(nlg(1):nlg(2))=(v_ideal(nlg(1):nlg(2))-2.5)*0.9;
v_ideal(nlg(3):nlg(4))=7.5;
v_up(nlg(3):nlg(4))=(v_ideal(nlg(3):nlg(4))+2.5)*1.1;
v_down(nlg(3):nlg(4))=(v_ideal(nlg(3):nlg(4))-2.5)*0.9;
v_ideal(nlg(5):nlg(6))=7.5;
v_up(nlg(5):nlg(6))=(v_ideal(nlg(5):nlg(6))+2.5)*1.1;
v_down(nlg(5):nlg(6))=(v_ideal(nlg(5):nlg(6))-2.5)*0.9;
v_ideal(nlg(7):nlg(8))=7.5;
v_up(nlg(7):nlg(8))=(v_ideal(nlg(7):nlg(8))+2.5)*1.1;
v_down(nlg(7):nlg(8))=(v_ideal(nlg(7):nlg(8))-2.5)*0.9;

%for loop runs through the points of the test
%checks that the speed is in the limits
%sets that the test is failed
%saves the failing points
%counts the failing points
for j=nlg(1):nlg(2)
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
        fail_points(j)=t(j);
        fail_count(i)=fail_count(i)+1;
    end
end
for j=nlg(3):nlg(4)
    if or(v(j)<v_down(j),v(j)>v_up(j))
        fail_vect(i)=1;
```

```
            fail_points(j)=t(j);
            fail_count(i)=fail_count(i)+1;
        end
    end
    for j=nlg(5):nlg(6)
        if or(v(j)<v_down(j),v(j)>v_up(j))
            fail_vect(i)=1;
            fail_points(j)=t(j);
            fail_count(i)=fail_count(i)+1;
        end
    end
    for j=nlg(7):nlg(8)
        if or(v(j)<v_down(j),v(j)>v_up(j))
            fail_vect(i)=1;
            fail_points(j)=t(j);
            fail_count(i)=fail_count(i)+1;
        end
    end

    %creates graph and saves it
    graph_t=strcat(testname(i));
    fig = figure('visible','off');
    plot(t,v,'--','color','b')
    hold on
    plot(t(nlg(1):nlg(2)),v_up(nlg(1):nlg(2)),'--','color','g')
    hold on
    plot(t(nlg(1):nlg(2)),v_down(nlg(1):nlg(2)),'--','color','g')
    hold on
    plot(t(nlg(3):nlg(4)),v_up(nlg(3):nlg(4)),'--','color','g')
    hold on
    plot(t(nlg(3):nlg(4)),v_down(nlg(3):nlg(4)),'--','color','g')
    hold on
    plot(t(nlg(5):nlg(6)),v_up(nlg(5):nlg(6)),'--','color','g')
    hold on
    plot(t(nlg(5):nlg(6)),v_down(nlg(5):nlg(6)),'--','color','g')
    hold on
    plot(t(nlg(7):nlg(8)),v_up(nlg(7):nlg(8)),'--','color','g')
    hold on
    plot(t(nlg(7):nlg(8)),v_down(nlg(7):nlg(8)),'--','color','g')
    hold on
    for k=1:size(n)
        if fail_points(k)~=0
            plot(fail_points(k),v(k),'x','color','r')
        end
    end
    title(graph_t);
    ylabel('Speed [km/h]');
    xlabel('Time [s]');
    legend('Speed','Upper limit', 'Lower limit','location','best')
    hold off
    figname=strcat(graph_t, '.fig');
    savefig(fig, fullfile(plotsdir_ex, figname));
    close (fig)

    %if no fail or failing for speed limit
    if or(fail_vect(i)==0,fail_vect(i)==1)
        %for loops calculate the standard deviation for each point
        %from the ideal speed
        for j=nlg(1):nlg(2)
```

```
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(3):nlg(4)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(5):nlg(6)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end
                for j=nlg(7):nlg(8)
                    if v_ideal(j)==0
                        std_low(j)=0;
                        std_dev(j)=0;
                    else
                        std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                        std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                    end
                end

                %calculates the mean among all the tested points of the
                %standard deviation
                ev_points(i)=nlg(8)-nlg(7)+nlg(6)-nlg(5)+nlg(4)-nlg(3)+nlg(2)-nlg(1)+4;
                testpoints=zeros(ev_points(i),1);
                testpoints(1:(nlg(2)-nlg(1))+1)=std_dev(nlg(1):nlg(2));
                testpoints((nlg(2)-nlg(1))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3))+2)=std_dev(nlg(3):nlg(4));
                testpoints((nlg(2)-nlg(1)+nlg(4)-nlg(3))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3)+nlg(6)-nlg(5))+2)=std_dev(nlg(5):nlg(6));
                testpoints((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5))+2:(nlg(2)-
nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7))+2)=std_dev(nlg(7):nlg(8));

                testpoints_low=zeros(nlg(8)-nlg(7)+nlg(6)-nlg(5)+nlg(4)-nlg(3)+nlg(2)-
nlg(1)+4,1);
                testpoints_low(1:(nlg(2)-nlg(1))+1)=std_low(nlg(1):nlg(2));
                testpoints_low((nlg(2)-nlg(1))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3))+2)=std_low(nlg(3):nlg(4));
                testpoints_low((nlg(2)-nlg(1)+nlg(4)-nlg(3))+2:(nlg(2)-nlg(1)+nlg(4)-
nlg(3)+nlg(6)-nlg(5))+2)=std_low(nlg(5):nlg(6));
                testpoints_low((nlg(2)-nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5))+2:(nlg(2)-
nlg(1)+nlg(4)-nlg(3)+nlg(6)-nlg(5)+nlg(8)-nlg(7))+2)=std_low(nlg(7):nlg(8));
                std_dev_tr(i)=mean(testpoints)^0.5;
                std_dev_low(i)=mean(testpoints_low)^0.5;
```

```
        end

    elseif sum(and(contains(test_tr,"POT"),contains(b,"POT")))==1

        nlt=zeros(2,1);
        nlg=zeros(2,1);
        %the points in the time serie with the closest longitude and
        %latitude values compared to fixed coordinates are taken
        [lt(1), nlt(1)]=min(abs(lt-lt_pot(1)));
        [lt(2), nlt(2)]=min(abs(lt-lt_pot(2)));
        [lg(1), nlg(1)]=min(abs(lg-lg_pot(1)));
        [lg(2), nlg(2)]=min(abs(lg-lg_pot(2)));

        %sets the ideal and upper and lower limit for speed according
        %to standards
        v_ideal(nlt(1):nlt(2))=40;
        v_up(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))+3)*1.05;
        v_down(nlt(1):nlt(2))=(v_ideal(nlt(1):nlt(2))-3)*0.95;

        %for loop runs through the points of the test
        %checks that the speed is in the limits
        %sets that the test is failed
        %saves the failing points
        %counts the failing points
        for j=nlt(1):nlt(2)
            if or(v(j)<v_down(j),v(j)>v_up(j))
                fail_vect(i)=1;
                fail_points(j)=t(j);
                fail_count(i)=fail_count(i)+1;
            end
        end

        %creates graph and saves it
        graph_t=strcat(testname(i));
        fig = figure('visible','off');
        plot(t,v,'--','color','b')
        hold on
        plot(t(nlt(1):nlt(2)),v_up(nlt(1):nlt(2)),'--','color','g')
        hold on
        plot(t(nlt(1):nlt(2)),v_down(nlt(1):nlt(2)),'--','color','g')
        hold on
        for k=1:size(n)
            if fail_points(k)~=0
                plot(fail_points(k),v(k),'x','color','r')
            end
        end
        title(graph_t);
        ylabel('Speed [km/h]');
        xlabel('Time [s]');
        legend('Speed','Upper limit', 'Lower limit','location','best')
        hold off
        figname=strcat(graph_t, '.fig');
        savefig(fig, fullfile(plotsdir_ex, figname));
        close (fig)

        %if no fail or failing for speed limit
        if or(fail_vect(i)==0,fail_vect(i)==1)
            %for loops calculate the standard deviation for each point
            %from the ideal speed
```

```
            for j=nlt(1):nlt(2)
                if v_ideal(j)==0
                    std_low(j)=0;
                    std_dev(j)=0;
                else
                    std_low(j)=(v_up(j)/v_ideal(j)-1)^2;
                    std_dev(j)=(v(j)/v_ideal(j)-1)^2;
                end
            end
            %calculates the mean among all the tested points of the
            %standard deviation
            ev_points(i)=nlt(2)-nlt(1);
            std_dev_low(i)=mean(std_low(nlt(1):nlt(2)))^0.5;
            std_dev_tr(i)=mean(std_dev(nlt(1):nlt(2)))^0.5;
        end
    %if the test name is not in the list of tracks to be tested, fail
    elseif contains(b, test_tr)==0
        fail_vect(i)=3;
    end

    %checks that there are not errors in the speed signal as in a
    %standard car would be impossible to have an acceleration higher
    %than 7km/h per millisecond
    for check_speed=1:n-1
        if v(check_speed+1)-v(check_speed)>7
            fail_vect(i)=4;
        end
    end

    %CHECK WHETHER THE TRY WAS A PASS OR A FAIL AND IF A PASS
    %ASSIGNEMENT OF THE MARK

    %mark_red array saves for each test a reduction of the mark based
    %on the number of failing points over the number of evaluated
    %points and if less than 10% saves it as a pass
    mark_red(i)=fail_count(i)/ev_points(i);
    if mark_red(i)<=0.1
        fail_vect(i)=0;
    end
    %each number assigned to fail_vect corresponds to an error occurred
    if fail_vect(i)==1
        pass_vect(i)="Fail for speed";
    elseif fail_vect(i)==2
        pass_vect(i)="Fail for overtime";
    elseif fail_vect(i)==3
        pass_vect(i)="Not enough data";
    elseif fail_vect(i)==4
        pass_vect(i)="Error in dataset";
    elseif fail_vect(i)==101
        pass_vect(i)="Exec not considered";
    elseif fail_vect(i)==0
        pass_vect(i)="Pass";
        if sum(v_ideal,'all')~=0
            if mark_red(i)<=0.02    %less than 2% fail points full mark
                mark_ex(i)=((std_dev_low(i)-std_dev_tr(i))/std_dev_low(i))*10;
            %between 2% and 5% fail points 20% reduction on mark
            elseif and(mark_red(i)>0.02,mark_red(i)<=0.05)
                mark_ex(i)=((std_dev_low(i)-std_dev_tr(i))/std_dev_low(i))*10*0.8;
            %between 5% and 10% fail points 40% reduction on mark
```

```
                elseif and(mark_red(i)>0.05,mark_red(i)<=0.1)
                    mark_ex(i)=((std_dev_low(i)-std_dev_tr(i))/std_dev_low(i))*10*0.6;
                end
            %while for the tracks with speeds to respects the mark is based
            %on the standard deviation, for the AV the mark is based on the
            %brake pedal presure taking 75 as 6/10 mark and 100 as a 10/10
            elseif sum(and(contains(test_tr,"AV"),contains(b,"AV")))==1
                mark_ex(i)=0.16*mean(max_br, 'all')-6;
            end
        end

        %updating loading percentage
        if perc_count<n_tr
%             figure(app.GoTAAppUIFigure)
            perc_count=perc_count+1;
            perc_ex=round(perc_count*100/n_tr,0);
            percentage_status=char(strcat("Execution ", string(perc_ex), "%"));
            app.statusLabel.HorizontalAlignment = 'left';
            app.statusLabel.Text = percentage_status;
        end
    end

    %CREATION OF SUMMARY TABLE AND SAVING

    ii=1;
    jj=1;
    mark_execution=strings(n_valid_tr, 3);
    while jj<n_tr
        %if pass or fail_vect is 1 or 2 copies the mark of execution
        if and(fail_vect(jj)>=0,fail_vect(jj)<3)
            if n_tries_tr(ii)==1
                mark_execution(ii,1:3)=[mark_ex(jj),"-","-"];
                ii=ii+1;
                jj=jj+1;
            elseif n_tries_tr(ii)==2
                mark_execution(ii,1:3)=[mark_ex(jj), mark_ex(jj+1),"-"];
                ii=ii+1;
                jj=jj+2;
            elseif n_tries_tr(ii)==3
                mark_execution(ii,1:3)=[mark_ex(jj), mark_ex(jj+1), mark_ex(jj+2)];
                ii=ii+1;
                jj=jj+3;
            end
        %if Langhe sets mark as "-"
        elseif fail_vect(jj)==101
            if n_tries_tr(ii)==1
                mark_execution(ii,1:3)=["-","-","-"];
                ii=ii+1;
                jj=jj+1;
            elseif n_tries_tr(ii)==2
                mark_execution(ii,1:3)=["-","-","-"];
                ii=ii+1;
                jj=jj+2;
            elseif n_tries_tr(ii)==3
                mark_execution(ii,1:3)=["-","-","-"];
                ii=ii+1;
                jj=jj+3;
            end
        %if not enough data, next test
```

```
        elseif fail_vect(jj)==3
            jj=jj+1;
        end
    end
    %create a table and a summary file
    mark_red=mark_red*100;
    report=table(testname(:),pass_vect(:),std_dev_tr(:),mark_ex(:),mark_red(:));
    report.Properties.VariableNames = {'Test name','Result','Standard
deviation','Mark','Error percentage'};
    writetable(report,execution_file);
    error=1;
end
```