POLITECNICO DI TORINO

Master's degree in Mechatronic Engineering

Master's Degree Thesis

Cylinder balancing control calibration in diesel engines

Politecnico di Torino

PUNCH | Torino

Supervisor
Prof. Stefano Malan
Co-Supervisors
Ing. Luca Fossati
Ing. Andrea Morgando
Ing. Stefano Coretti

Candidate
Pietro Sansone 286127

Academic Year 2021-2022

“Fatti non foste

a viver come bruti,

ma per seguir

virtute e canoscenza”

Abstract

Automotive control systems have become a driving factor in automotive innovation over the last thirty years. In order to meet the enhanced requirements for lower fuel consumption, lower exhaust emissions, improved safety as well as comfort and convenience functions, automotive control had to be applied.

Nowadays the interest toward fuel consumption reduction in reciprocating internal combustion engines has achieved a key role starting from the first energy crisis during the 70's. Even if in alternate phases, such interest had further increased during the following years assuming a fundamental role in the last years [1], these concerns grew simultaneously with the interest towards drivability improvement which has become a key factor in modern cars. Drivability is defined as the degree of smoothness and steadiness of acceleration of an automotive vehicle and strongly depends on the engine behavior. Since the engine is responsible for producing the torque that pushes the vehicle forward it is a major player in determining drivability performance indexes.

In an engine, torque production is the direct consequence of gases expansion inside the combustion chamber as a result of fuel combustion and it is not evenly distributed over a crankshaft revolution. For example, in a 4 cylinders engine, peaks of torque will occur each 180° of crankshaft revolution, for a 6 cylinders engine each 120°. Ideally, the function that represents torque profile over a crankshaft revolution could be taught as a sinusoid with different frequencies depending on the number of cylinders.

To meet NVH (Noise Vibration Harshness) requirements, in multi cylinders engines, it is paramount important to have the same torque produced in each cylinder, otherwise the engine would be running faster at some time and slower at others, producing vibrations which degrade drivability indexes. The speed with which the piston travels down along the cylinder, and consequently how fast the crankshaft spins, is a direct consequence of the pressure profile generated inside the combustion chamber. Moreover, the pressure profile strongly depends on the amount of fuel injected and how it is injected.

The engine component that is directly responsible for fuel injection is the injector. Over the years several designs of injectors have been adopted, starting from purely mechanical injectors getting to the most advanced electro mechanical ones. With the advances in electronics and digital technologies in the 70's, it became feasible to electronically control the fuel injections to increase the fuel combustion efficiency and at the same time reduce emissions. The problem with cost efficient electronic fuel-injection systems is the need of periodic calibration of the cylinder-wise fuel injections. Without calibration, the amounts of fuel injected into the cylinders deviate significantly [2]. This deviations, commonly called drifts, are mainly produced by the aging of the injectors or manufacturing errors and are the main cause of degradation of the NVH performances.

Knowing what is the cause of injection errors and their consequences, it is clear how a control system that manages the injections to equalize cylinder-wise torque contributions is needed. This control system takes the name of “Cylinder Balancing”, also known as CB, it acts on the amount of fuel injected by evaluating the Engine Unbalance from flywheel signals measures. Depending on the type of engine, the CB needs to be calibrated accordingly, this activity requires to be performed by means of tests on vehicle because of the strong dependance of CB behavior from the driveline. It follows that all the tests that need to be performed in order to calibrate this control system are highly cost demanding, since the complete vehicle is required at the roller test bench facility. Moreover, in the calibration procedure under consideration it is required a very high amount of hours on vehicle in order to get to a final calibration.

The aim of the thesis work is to analyze the current calibration procedure in order to detect areas of improvement in the calibration effort and standardization. The main goal of the reviewed calibration procedure is to limit the tests at the roller test bench together with the workload to be exploited at the desk by the calibrator while keeping the same performances of the control system. Several steps have been taken to develop and validate the new calibration procedure, at first the current procedure was studied and the area of improvements were highlighted. For some calibration procedures it was possible to virtualize the calibration by retrieving results based on available information instead of new tests, for others it was built a simulation environment on Simulink, able to reproduce the engine response relative to signals of interests, avoiding the need to perform new tests at the Dyno. Some of the calibration procedures could not get virtualized in their whole, meaning that tests at the Dyno are the only way to get the information needed to deliver a calibration. In these situations, calibration tools were developed in Python, whose purpose is to assist the calibrator in order to reduce the manual effort and consequently the total time of calibration.

Acknowledgements

Questo elaborato conclude un percorso durato 5 anni, oggi ho raggiunto uno dei traguardi più importanti per me, fino ad ora.

Bisogna dire che durante gli anni del politecnico la mia strada si è incrociata con quella di numerose persone, alcune hanno lasciato il segno, dando un contributo importantissimo ai fini del raggiungimento di questo obiettivo, per questo motivo in queste poche righe ritengo necessario porgere a loro un ringraziamento.

In primo luogo voglio ringraziare il Professor Malan, l'Ing. Fossati, l'Ing. Morgando e l'Ing. Coretti i quali sempre estremamente disponibili durante la stesura della tesi, sono stati determinanti per la buona riuscita di questo elaborato.

Ringrazio i miei amici di sempre, con i quali ho condiviso la maggior parte della mia vita, oltre che questi anni di università, Ale, Ema, Patru, Pole, Ste che da quando ho ricordi è stato come un secondo fratello per me, Sandro, mio coinquilino dal TIL alla laurea magistrale, anche se con qualche interruzione tra pandemie globali e Exchange.

Ringrazio Giulia, la mia ragazza, con la quale ho condiviso l'ultimo anno e mi ha sempre sopportato e supportato durante gli esami e soprattutto ha sopportato i numerosi "no" quando dovevo studiare.

Infine, ringrazio le persone che hanno fatto sì che io possa essere qua in questo momento a scrivere queste parole e conseguire questo titolo, coloro che mi hanno sostenuto sotto ogni aspetto e messo sempre nella condizione migliore per poter provare a realizzare i miei sogni. Non mi avete mai fatto mancare nulla.

Grazie Mamma, Papà, Matteo e Nonni tutti, questo traguardo è soprattutto il vostro.

Contents

1 Introduction

1.1 Motivation

Automotive control systems have become a driving factor in automotive innovation over the last thirty years. In order to meet the enhanced requirements for lower fuel consumption, lower exhaust emissions, improved safety as well as comfort and convenience functions, automotive control had to be applied. Nowadays the interest toward fuel consumption reduction in reciprocating internal combustion engines has achieved a key role starting from the first energy crisis during the 70's. Even if in alternate phases, such interest had further increased during the following years assuming a fundamental role in the last years [1]. Simultaneously, interest towards drivability improvements has become a key factor in modern cars. Drivability is defined as the degree of smoothness and steadiness of acceleration of an automotive vehicle, since the engine is the responsible of torque production through a combustion process it is clear how this process has a key role in determining drivability indexes.

In multi cylinder engines torque production varies over the crankshaft revolution and it depends on the number of cylinders considered, for example, in a 4 cylinders engine, peaks of torque will occur each 180° of crankshaft. Torque production is the direct consequence of the expansion of gasses inside the combustion chamber as a consequence of fuel combustion. The pressure generated in the combustion pushes the piston downwards and its linear motion is converted in rotational motion through the piston rod connecting it to the crankshaft, the ideal speed profile assumed by the crankshaft can then be thought as a sinusoid.

To meet NVH (Noise Vibration Harshness) requirements is paramount important to avoid having Torque distribution that is not evenly spread over a crankshaft revolution, this would lead to have the engine running faster at some time and lower at others, producing vibrations. The speed with which the piston travels down in the combustion chamber and consequently how fast the crankshaft spins is a direct consequence of the Pressure profile generated in the combustion chamber. The pressure profile strongly depends on the amount of fuel injected and how it is injected. In diesel engines fuel is injected following Injection patterns which means that depending on the situations fuel is injected at different steps and amounts.

The component that is directly responsible for fuel injection is the injector. Over the years several designs of injectors have been adopted, starting from purely mechanical injectors getting to the most advanced electro mechanical ones. With the advances in electronics and digital technologies in the 70's, it became feasible to electronically control the fuel injections to increase the fuel combustion efficiency and at the same time reduce emissions. The problem with cost efficient electronic fuel-injection systems is the need of periodic calibration of the cylinder-wise fuel injections. Without calibration, the amounts of fuel injected into the cylinders deviate significantly [2]. This deviations, commonly called drifts, are mainly produced by the aging of the injectors or manufacturing errors.

Knowing what is the cause of injection errors and their consequences, it is clear how a control system that manages the injections to equalize cylinder-wise torque contributions is needed. This control system takes the name of "Cylinder Balancing". This acts on the amount of fuel injected by performing corrections on injectors' energizing times, evaluating the balance of the engine from flywheel signals measures. This control system has to be adapted depending on the type of engine it is used on, this adaptations are called calibrations.

The calibration of the Cylinder Balancing control system requires many testing activities on the roller test bench, these, are time and cost demanding. Time needed for calibration and number of tests if reduced would lead to a great advantage in the calibration process. This thesis work responds to this need by developing alternative calibrating procedure with the support of tools developed in python environment.

1.2 Objectives

Scope and objective of the thesis work is to develop calibrating procedures with an eye on time and tests reduction following the steps:

- Research and assesses the current state of the art of the current calibrating procedures;
- Identify the calibrations requiring much time and tests;
- Developing alternative calibration procedures;
- Comparing the performances obtained with the original calibration procedures with the reviewed one;

2 Background

2.1 Purpose of Fuel Injection Systems

The performance of diesel engines is heavily influenced by the injection system design. In fact, the most notable advances achieved in diesel engines resulted directly from superior designs of this component. While the main purpose of the system is to deliver fuel to the cylinders of a diesel engine, it is how that fuel is delivered that makes the difference in engine performance, emissions, and noise characteristics.

Unlike its spark-ignited counterpart, the diesel fuel injection system delivers fuel under extremely high injection pressures. This implies that the system component design and materials should be selected to withstand higher stresses in order to perform for extended durations that match the engine durability targets. Greater manufacturing precision and tight tolerances are also required for the system to function efficiently. In addition to expensive materials and manufacturing costs, diesel injection systems are characterized by more intricate control requirements. All these features add up to a system whose cost may represent as much as 30% of the total cost of the engine.

In order for the engine to effectively make use of the fuel provided by the injection system:

i. Fuel must be injected at the proper time, that is, the injection timing must be controlled

ii. The correct amount of fuel must be delivered to meet power requirement, that is, injection metering must be controlled

However, it is still not enough to deliver an accurately metered amount of fuel at the proper time to achieve good combustion. Additional aspects are critical to ensure proper fuel injection system performance including:

- *Fuel atomization*. It ensures that fuel atomizes into very small fuel particles and it is a primary design objective for diesel fuel injection systems. Small droplets ensure that all the fuel has a chance to vaporize and participate in the combustion

process. Any remaining liquid droplets burn very poorly or are exhausted out of the engine. While modern fuel injection systems are able to produce fuel atomization characteristics far exceeding what is needed to ensure complete fuel evaporation during most of the injection process, some injection systems designs may have poor atomization during some brief but critical period of the injection phase.

- *Bulk mixing.* While fuel atomization and complete evaporation of fuel is critical, ensuring that the evaporated fuel has sufficient oxygen during the combustion process is equally as important to ensure high combustion efficiency and optimum engine performance. The oxygen is provided by the intake air trapped in the cylinder and a sufficient amount must be entrained into the fuel jet to completely mix with available fuel during the injection process ensuring complete combustion.

- *Air Utilization.* Effective utilization of the air in the combustion chamber is closely tied to bulk mixing and can be accomplished through a combination of fuel penetration into the dense air that is compressed in the cylinder and dividing the total injected fuel into a number of jets. A sufficient number of jets should be provided to entrain as mush of available air as possible while avoiding jet overlap and the production of fuel rich zones that are oxygen deficient.

The main purposes of the diesel fuel injection systems are graphically represented in Figure 2.1.1.

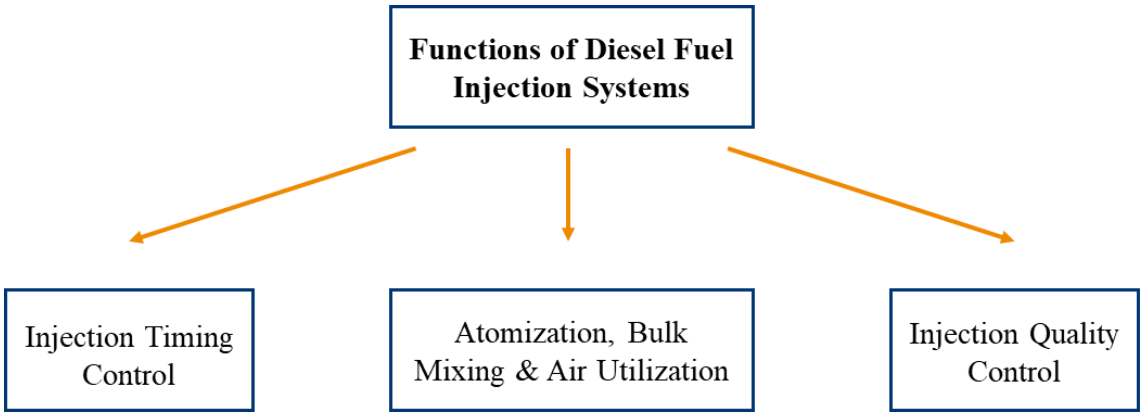


Figure 2.1.1: Main Functions of Diesel Fuel Injection Systems

2.2 Basic Fuel System Components

With a few exceptions fuel systems can be broken down into two major component groups:

- ***Low pressure side components*** serve to safely and reliably deliver fuel from the tank to the fuel injection system. These components include the fuel tank, fuel supply pump and the fuel filter.

- ***High Pressure side components*** that create high pressures, meter and deliver the fuel to the combustion chamber. They include the high pressure pump, the fuel injector and fuel injector nozzle, common rail systems also include an accumulator.

Fuel injection nozzles can be categorized as hole-type or throttling pintle type and as either as closed or open. Closed nozzles can be actuated hydraulically using a simple spring-biased mechanism or using servo control. Open nozzles as well as some newer closed nozzle injector design can be directly actuated.

Metering of the Injected fuel amount is commonly carried out in either the high pressure pump or the fuel injector. A number of different fuel metering approaches exists including: pressure metered at constant time interval (PT), time metered at constant pressure (TP) and time/stroke metered (TS).

Most fuel injection systems use electronics to control the opening and closing of the nozzle. Electric signals are converted into mechanical forces using some type of actuators. Commonly, these actuators can be either electromagnetic solenoids or active materials such as piezoelectric ceramics.

2.3 Common Diesel Fuel Injection System Architectures

There are three common architectures of diesel fuel injection systems:

- **Pump-Line-Nozzle**
- **Unit Injector**
- **Common Rail**

The *Pump-Line-Nozzle* (PLN) architecture uses a central injection pump driven by the engine geartrain, Figure 2.3.1. The injection pump feeds separate injection nozzles located in the cylinder head above each cylinder. Lines, which must be of exactly equal length, link the pump with the nozzles. In the case of an *in-line pump,* the central pump incorporates a number of separate plunger/barrel pumping elements (such as that shown in Figure 2.3.1), each serving one injector, that combine the high pressure generation and

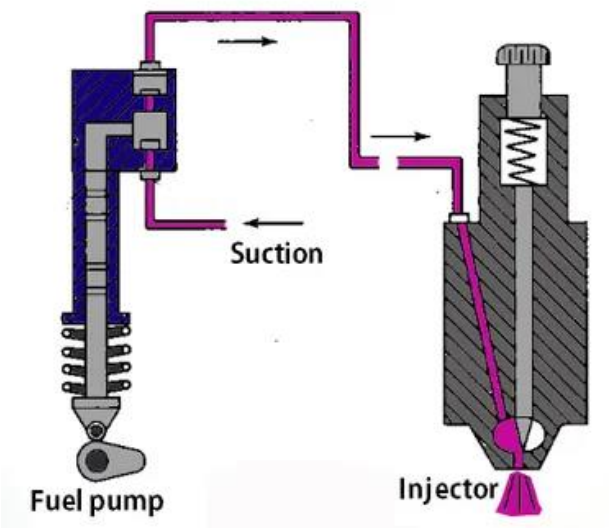


Figure 2.3.1: Pump-Line-Nozzle

metering functions. Each nozzle incorporates a needle valve and the orifice which provide fuel atomization. The PLN fuel systems used to be the most common type of diesel injection system, dominating most diesel engine applications.

In addition to the in-line pump design, where each injector is fed by a separate pumping element, several other configurations that have been developed, including the distributor/rotary pump system, provide one pumping element that serves a number of cylinders.

The *Unit Pump* (UP) system, a variation of the PLN system, also incorporated individual camshaft-driven injection pumping elements for each cylinder. However, rather than incorporating the pumping elements into one central pump, each element is contained in a separate pump located close to the cylinder it serves. The injection nozzle and pump are connected by a shorter fuel line than the PLN.

The *Unit Injector* (UI) architecture incorporates the high pressure pumping element, fuel metering and the injector in one device. While there are many variations of the unit injector design, one common design shown in Figure 2.3.2 effectively incorporates the UP system components into one unit with the added benefit of eliminating the high pressure line. The elimination of the injection line decreases the possibility of wave superposition, which may lead to secondary injections and contribute to injection delays. Traditionally, unit injector systems have been able to deliver extremely high injection pressures.

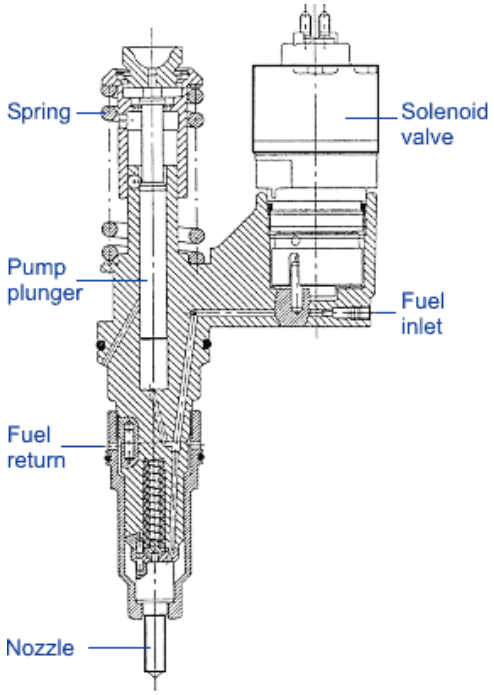


Figure 2.3.2: Electronic Unit Injector

The latest and most adopted injection system is the *Common Rail* architecture. It employs a common pressure accumulator, called the rail, which is mounted along the engine block, Figure 2.3.3. The rail is fed by a high pressure fuel pump that is driven by the crankshaft. High pressure injection lines connect the common rail to the fuel injectors. In the mechanical incarnation of the common rail, the injectors are actuated by overhead cam or rocker arm mechanisms. In modern, electronically controlled systems, such as that in Figure 2.3.3, the injectors are actuated by electronically actuated valves. This architecture allows to have a better controllable pressure thanks to the rail which acts as a reservoir and stabilizes the pressure at which the fuel is delivered over the opening of the nozzle leading to a more precise metering of the injected fuel amount.

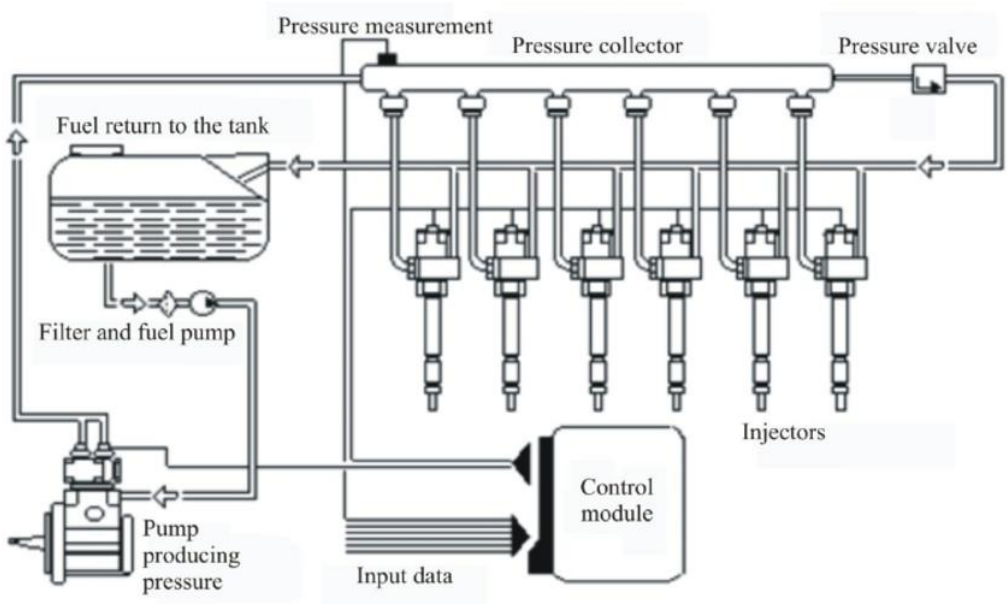


Figure 2.3.3: Common Rail System

Pressure inside the rail depends on the crankshaft speed, it means that the pressure inside the rail is not always equal to its maximum. This is favorable when operating at low rpm and load, indeed, in this situation a small amount of injected fuel is required, having the highest pressure possible in the rail when such a small amount of injected fuel is required would mean having extremely small Energizing Times for the injectors. This could lead to a very high error in the injected fuel quantity coming from a small error in the energizing time.

Peak injection pressures for some high-speed Direct Injection (DI) diesel engines have been maintained at about 200 MPa for the first decade of the 21st century. Beneath this seeming stagnation is an important transition in fuel injection technology. In the last years of the 20th century, the highest injection pressures of about 200 MPa for light-duty applications were soon phased out in favor of the increased flexibility in multiple injections and the independence of injection pressure from engine speed offered by common rail systems. The highest peak pressures available from light-duty common rail injection systems only reached 200 MPa towards the end of the first decade of the 21st century and will continue to evolve to higher levels.

Common rail systems for light-duty applications, such as Bosch's Hydraulically Amplified Diesel Injector (HADI) that can reach pressures of 250 MPa and Denso's 4th generation system that can reach 300 MPa have been under development.

2.4 Definition of Terms

In this section the terms adopted to describe the injection process for common rail systems and previously adopted systems are illustrated, these last differ from the first just in some definition which depends on the actuation system.

2.4.1 Common Rail Systems

- ***Nozzle*** refers to the part of the nozzle body/needle assembly which interfaces with the combustion chamber of the engine. Terms like P-Type, M-Type, or S-Type nozzle refer to standardized dimensions of nozzle parameters, as per ISO specifications, Figure 2.4.1.

Figure 2.4.1: Diesel Injector Nozzle

- ***Nozzle holder*** or ***Injector body*** refers to the part the nozzle is mounted on. In conventional injection systems this part mainly served the nozzle mounting and nozzle needle spring preloading function. In common rail systems, it contains the main functional parts: the servo-hydraulic circuit and the hydraulic actuator (Electromagnetic or piezoelectric).

- ***Injector*** commonly refers to the nozzle holder and nozzle assembly Figure 2.4.2.

Figure 2.4.2: Chevy Duramax Diesel Injector

- ***Start of Energizing*** and ***Start of Injection***, these are usually expressed in crank angle degrees (CAD) relative to Top Dead Center (TDC) of the compression stroke. Start of Energizing (SOE) is indicated by a measured parameter such as the time at which an electric actuation signal is sent to the injector. Due to the mechanical response of the injector, an "injector lag" exists, which leads to a delay between the moment at which the electric signal is sent to the injector (SOE) and the instant at which the fuel actually exits the injector nozzle, referred as Start of Injection (SOI).

- ***End of Injection*** *(EOI)* is the time in the cycle when fuel injection stops.

- ***Injected Fuel Quantity*** is the amount of fuel delivered to a cylinder per power stroke. It is often expressed in $mm^3/stroke$ or $mg/stroke$.

- ***Injection duration*** is the period of time during which fuel enters the combustion chamber from the injector. It is the difference between the EOI and SOI and is related to injection quantity.

- ***Energizing Time (ET)*** is the time during which the injector is energized, meaning the nozzle is delivering fuel to the combustion chamber.

- ***Injection Pattern*** the rate of injection of fuel often varies during the injection period. Figure 2.4.3 shows three common injection rates over time: boot, ramp and square. The difference between injection patterns lays in the injector opening and closing rate which is represented by the derivative of the injection rate.

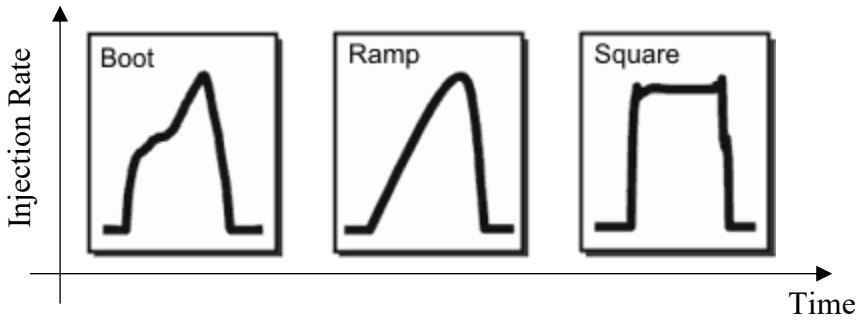


Figure 2.4.3: Common Injection Rate Shapes

- ***Multiple Injections Events*** Figure 2.4.4, current diesel engines employ multiple fuel injections. Multiple injections are divided in:

1) Pre-Injections
2) Main-Injection
3) Post-Injection
4) After-Injection

Pre-injections are small Injections (Compared to the main) which lead to a slight increase of the pressure inside of the combustion chamber (low torque produced) reducing the noise when the main injection takes part. Moreover, these small injections heat up the combustion chamber so that when the Main-Injection occurs it burns with higher efficiency and with less pollutant emission. The main injection, which is responsible for torque production, is followed by Post-Injections which aim is to reduce the production of soot. Soot is the result of a not complete combustion of the fuel, after the Post-injections the amount of soot is reduced.

After-Injections are responsible for heating up the exhaust gasses and consequently the Exhaust Gasses Treatment (EGT) devices such as DPF, DOC, SCR, in this way they are in their working temperature window.

After-Injections are divided in multiple pulses instead of a big one, in order to avoid the problem of high pressure waves which would lead to some fuel leakage into the oil pan, however, dividing the post injections into smaller ones has the problem of metering the amount of fuel injected after the first pulse due to the pressure wave deriving from previous pulses.

The number of pulses may vary depending on the so called “Combustion mode” decided for a specific working situation of the engine. Supposing for example to start the engine in cold condition, the combustion mode may be set to Warm Up

and for sure the after pulses will be present in order to warm up quickly the after treatment devices, once they are warm enough the engine could move into Warmed Up mode, which eliminates the after pulses that are no more needed. There are other several combustion modes that depend on particular situations, the previous one were cited as they are the most important and can help clearing the ideas.

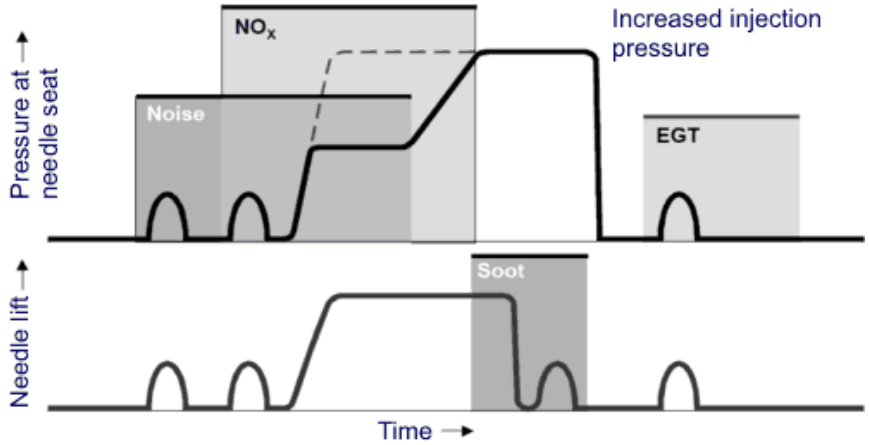


Figure 2.4.4: Multiple injections purposes

- ***Injection Pressure*** it is referred to the mean pressure in the rail.

2.4.2 Pump-Line-Nozzle and Unit-Injector Systems

- ***Start of Delivery*** in these systems, fuel injection is coordinated with the generation of high pressure, SOE is no longer adopted, instead, Start of Delivery is adopted. SOD is the time when the high pressure pump starts to deliver fuel to the injector. The difference between the Start of Delivery and SOI is affected by the length of time it takes for a pressure wave to travel between the pump and the injector, this depends on the length of the line and the speed of sound in the fuel.
- ***Injection Pressure*** it refers to the maximum pressure during an injection (peak injection pressure).

2.5 Injectors drift

Multiple injections lead to several improvements as described in the previous section. To fully achieve the mentioned improvements from multiple injection techniques, the proper hydraulic response of the injector is crucial, as these techniques largely rely on the correct injection timing and temporal space between injections (dwell time), as well as the

amount of fuel injected. Moreover an optimal metering of the injected fuel quantity allows to better control the torque delivery of the engine as a direct consequence [5].

Consequently, variations of the hydraulic performance of an injector have become a matter of interest, including alterations due to the aging of the injectors. In particular, several authors have underlined variations in the injection process due to the presence of deposits in the outlet orifices, which could appear due to nozzle geometry, fuel composition, high temperatures or manufacturing defects [6].

The rate of injection at the beginning of the injector lifetime is compared to that provided after aging during main injection Figure 2.5.1. It is noticed that aging led to a decrease of the injected fuel mass of about the 6.2% [5]. Another source of drift may be manufacturing defects, leading in having non nominal injectors sizes.

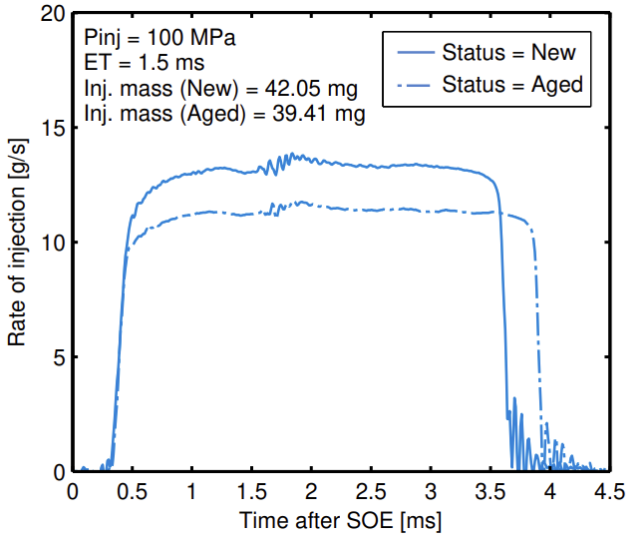


Figure 2.5.1: Rate of Injection measurement before and after aging [5]

At this point it is clear that when concerning about torque delivery, injector drift due to aging or manufacturing cannot be neglected. Indeed, when trying to balance the torque produced by each cylinder the difference between the expected injected quantity and the actual injected quantity has to be compensated, this compensation is performed by the so called Cylinder Balancing Control System.

3 Cylinder Balancing

The fuel quantity to be injected is calculated by the engine management system which translates the fuel quantity into energizing time which is then fed to the injector. It has to be said that the injection time is divided in multiple injections times following the injection pattern. Dividing the primary injection time leads in having very short injection times, errors due to varying injector rise and fall times thus introduce a relatively large fuel mass error. Another source of errors is the injection pressure difference Δp between the measured rail pressure and the actual. At the first time portion of the injection procedure, the pressure in the fuel rail is decreased at the location of the injector, resulting in pressure oscillations. At subsequent injection time portions, the pressure difference Δp is no longer at its nominal value, contributing to an error of the injected fuel mass m_f. Additional errors may be caused by clogging of the injector nozzles.

Differences in the injection quantity, from the nominal value, perturb the response of the engine. More in depth, considering the system composed by the crankshaft, piston rods and pistons, different fuel injections inside the combustion chamber produce different pistons linear speeds and consequently different crankshaft rotational speeds at same loads. It follows that to determine whether the different cylinders in an engine are balanced is possible by measuring the crankshaft speed. In this chapter the approach with which fuel injection errors are detected and compensated it is described.

The approach presented in the following section exploits signals coming from the crankshaft, especially from the phonic wheel, and uses them in order to determine which is the unbalanced cylinder and how much it is unbalanced in order to act with a coherent correction.

3.1 Crankshaft position sensing

Determining which is the angular position of the crankshaft is crucial in order to detect the unbalance and act with a correction on the cylinder that is responsible for the production of the unbalance.

The technology adopted for the measurement of the crankshaft position and consequently its speed, is based on the interaction between a sensing element Figure 3.1.1 and teeth that are present on the so called phonic wheel, an example can be seen in Figure 3.1.2 part 24.

The phonic wheel can be thought of as a gear with 60 equally spaced teeth, with two teeth removed, creating the sync gap. Crank sensor data result in the reading of 58 falling edges

which are 6 degrees spaced. The synch gap is needed to avoid a drift in the reading, when the largest gap, corresponding to the sync gap, is read, the crankshaft has for sure completed a round and the system can be resynchronized. The tooth are numbered from 1 to 58, the first is always the one after the sync gap. From these kind of measurements the engine speed can be retrieved.

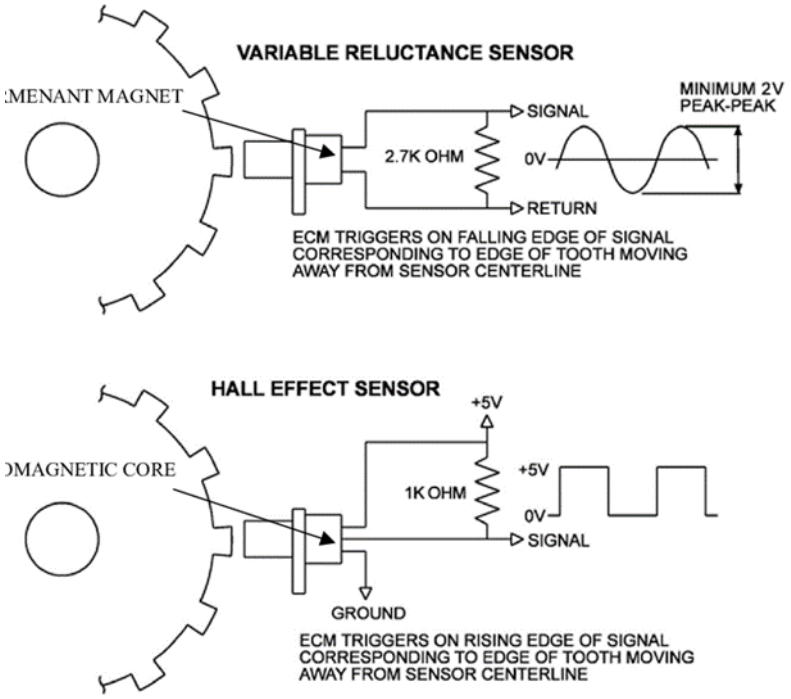


Figure 3.1.1: Sensing elements

It is now possible to know, for example, what is the time needed from one falling edge to another, knowing that each edge is 6° spaced, that is the time to cover that angular distance and the engine speed can be retrieved. This time is called T6 by convention and it will be a crucial information consumed by the cylinder balancing control.

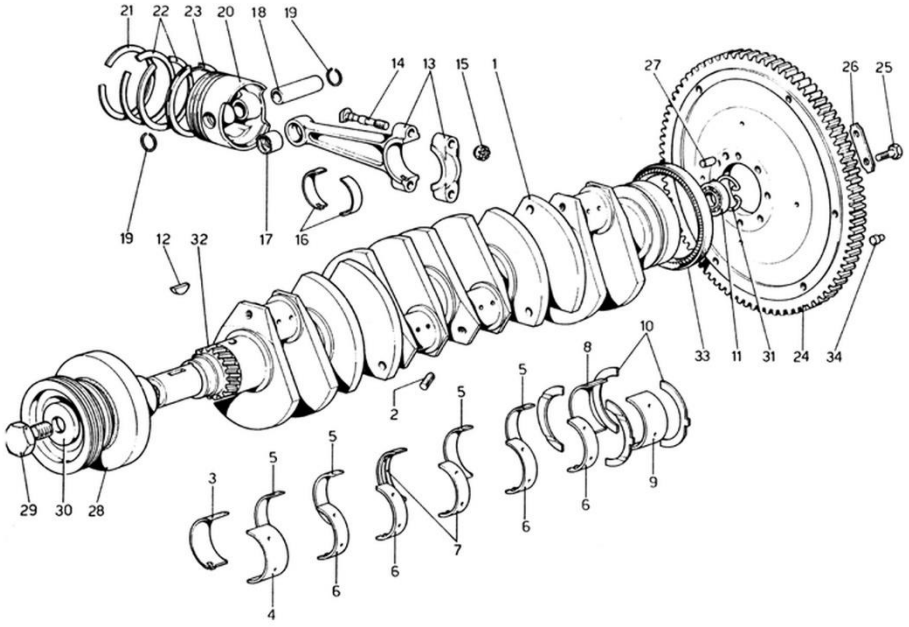


Figure 3.1.2: Crankshaft

3.2 Unbalance Estimation

The engine speed is retrieved from the crank wheel sensor, ideally, in a perfectly balanced engine at steady state operation the crankshaft speed profile assumes a shape similar to that presented in Figure 3.2.1, for a 4 cylinders engine. The sinusoidal shape of the engine speed derives from the fact that the combustion process is not homogeneously distributed over the crankshaft revolution, the IMEP (Internal Mean Effective Pressure) inside the combustion chamber is not constant. Moreover, in transient conditions, to the sinusoidal trend an increasing/decreasing trend depending if the engine is accelerating or decelerating is added. The engine speed in acceleration would assume the shape reported in Figure 3.2.2.

The engine speed is not directly measured but it is retrieved by the phonic wheel signal, measuring the time in between each falling edge. As it was said in the previous section the phonic wheel has 60 teeth, this means that a falling edge is detected each 6 degrees of crankshaft rotation, for convention the time in between two falling edges is called T6 which is simply the time required to turn the crankshaft of 6 degrees, after a crankshaft revolution an array of 60 T6 is then recorded.

In order to understand the concepts that will follow it is paramount important to highlight a convention that is used to refer to phenomena happening inside the engine which are referred through the engine orders. The engine orders can be resumed as follows:

- **Order 0.5**: Phenomena happening once each engine cycle (2 revolutions)
- **Order 1**: Phenomena happening twice each engine cycle
- **Order 1.5**: Phenomena happening 3 times each engine cycle and so on…

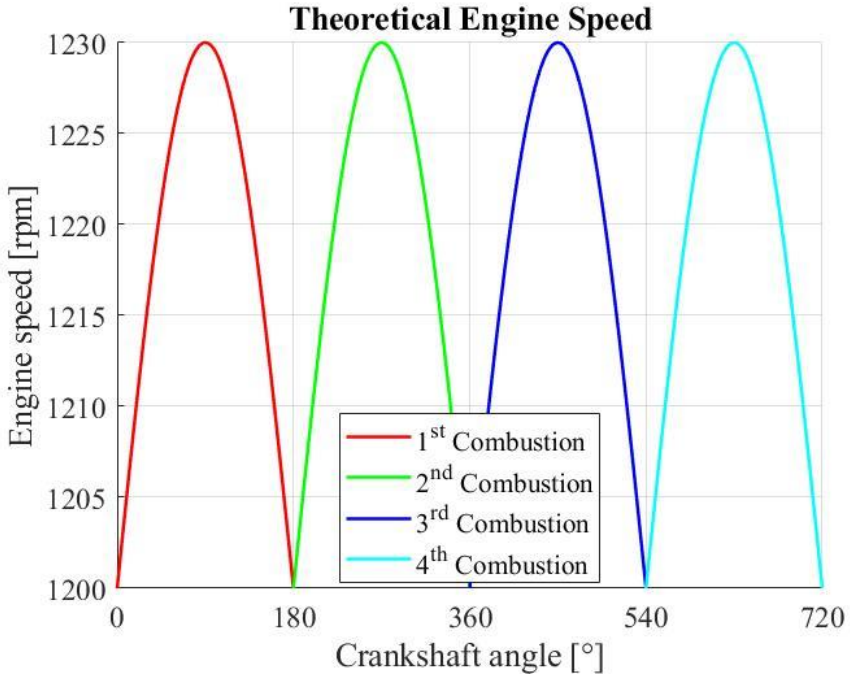


Figure 3.2.1: Theoretical engine speed.

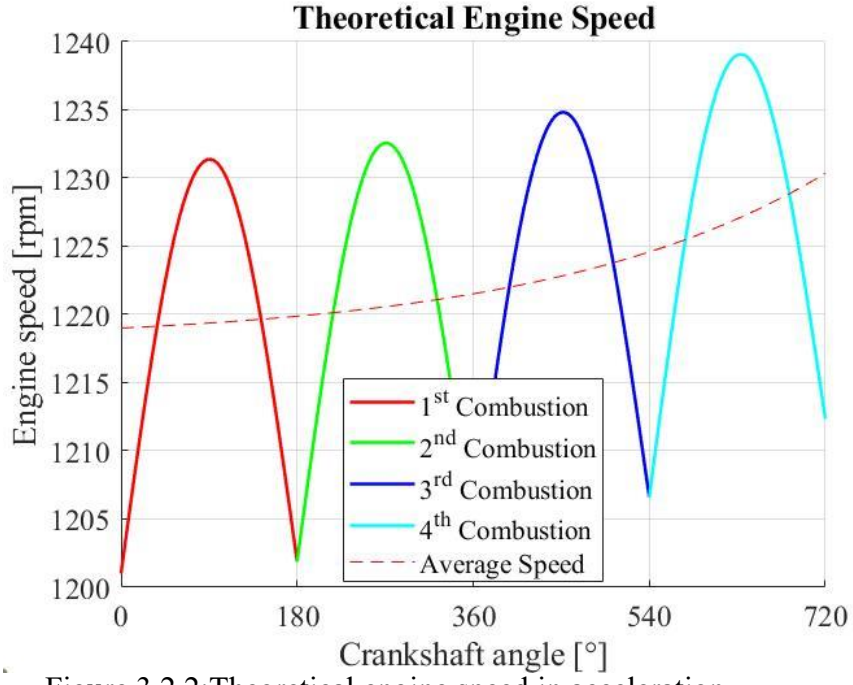


Figure 3.2.2:Theoretical engine speed in acceleration

In order to fix the ideas the following example can be taken into account. The first phenomena that comes into mind when thinking about an engine cycle is the combustion,

this process in a 4 cylinder engine happens four times each engine cycle (2 revolutions) this means that the combustion phenomenon has an order = 4/2 =2.

According to Nyquist theorem in order to measure a phenomenon without affecting its spectrum, it must be sampled at least at twice its frequency. This means that to obtain the speed profile reported in Figure 3.2.1 the engine speed must be sampled at least with an order 4 (8 times each engine cycle). Order 4 corresponds to sampling each 90 degrees of crankshaft revolution which in terms of time corresponds to a T90 (i.e. time required to perform 90 degrees crankshaft revolution). In a 4 cylinders engine, the engine speed signal shall be reduced to a sequence of T90 samples, filtered in order to show only relevant orders (i.e. 0.5, 1, 2).

In a perfectly balanced 4 cylinders engine, if the spectrum of the signal is analyzed, through its fast Fourier transform, the only component should be the one of order 2. If two cylinders are unbalanced the order 1 would appear Figure 3.2.3, if just one unbalanced cylinder is present an order 0.5 would then appear Figure 3.2.4.

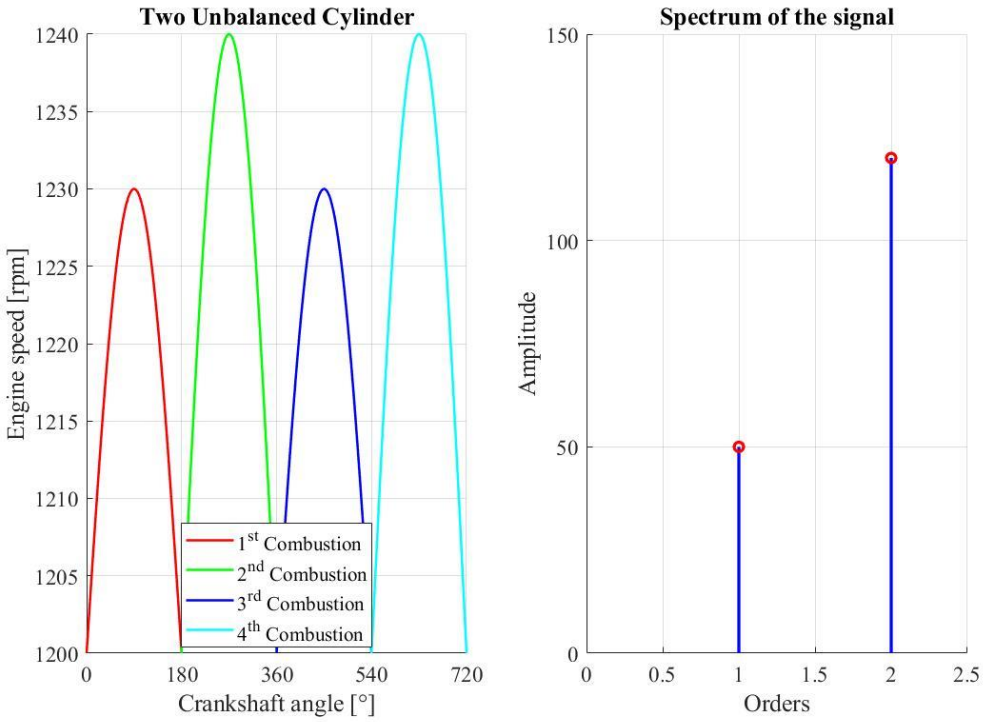


Figure 3.2.3: Two unbalanced cylinders and their spectrum

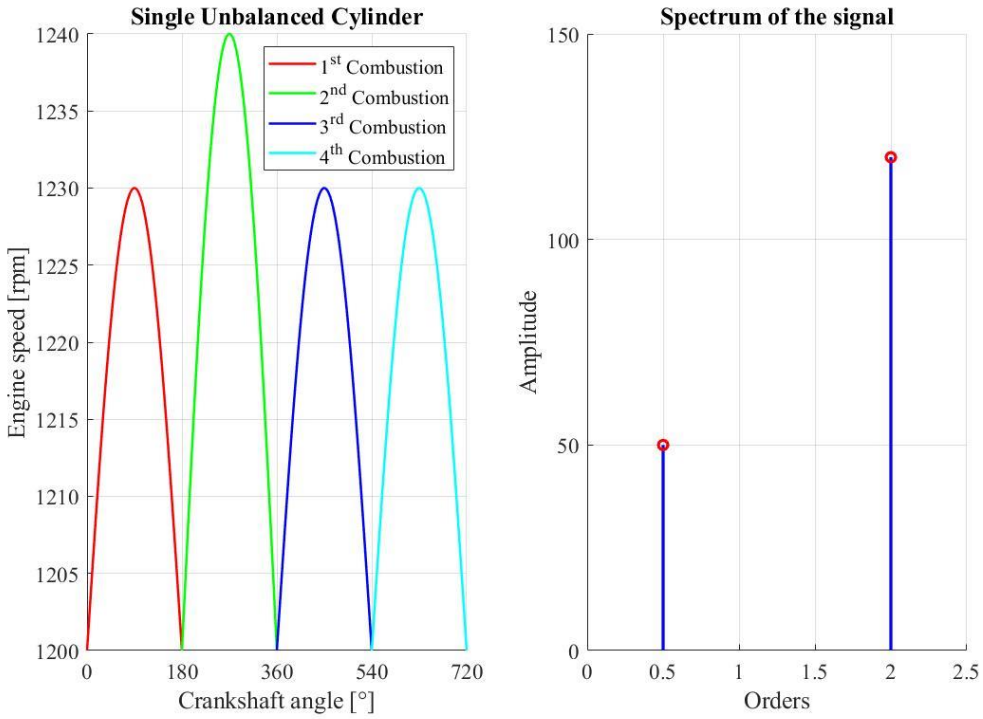


Figure 3.2.4: Single unbalanced cylinders and its spectrum

3.2.1 CBE System

CBE stands for Cylinder Balance Estimator, this is a procedure which allows to detect an unbalance and its magnitude, providing the information to the Cylinder Balancing (CB) control which will later manage the information provided and correct the system Figure 3.2.5.

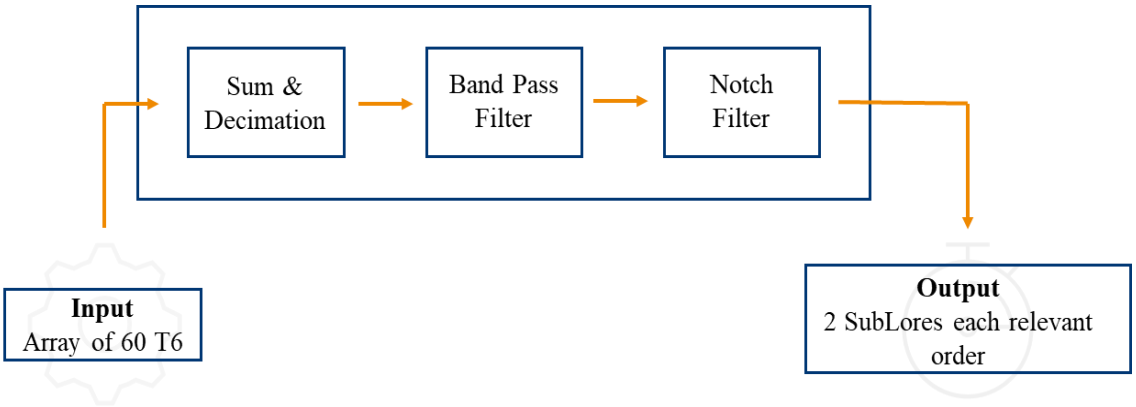


Figure 3.2.5: CBE Operation

Now it is clear that for the scope of cylinder balancing the smallest relevant order is the 0.5. In order to observe such a phenomena in a 4 cylinders engine, in which a firing event

happens each 180 crankshaft degrees, according to Nyquist one should sample the phonic wheel signal at least each 90° of crankshaft

CBE consumes an array of 60 T6, once again, a T6 is the time required to detect two consecutive teeth on the phonic wheel. The array of 60 samples is later reduced to an array of two T90 samples through sum and decimation. This action is exploited in order to reduce the computational load of the ECU, since for the sake of cylinder balancing in a 4 cylinders engine T90 samples are already satisfying (i.e. the spectrum is not impacted and the computational load is reduced enough). A schematic resuming how sum and decimation works is reported in Figure 3.2.6, at first the 60 T6 samples are summed in groups of 5 samples forming 12 groups of T30 samples. After the first grouping is performed, it is needed to apply a anti – aliasing filter to the T30 samples in order to remove the superimposed noise that was found to appear after the first operation.

Once the filtering is completed the samples are finally grouped in two T90, these samples are called Sub-Lores C which summed together form the Lores C period, the C stands for combustion, meaning that a Lores C is the time in between two consecutive firing events. The notation Lores is generally used to refer to a period of time in between two consecutive events, while the Sub-Lores is half of the Lores.

After sum and decimation, the samples pass through a band pass filter which let pass only frequencies around relevant orders, moreover if the band pass filter did not manage to eliminate unwanted noises very close to the order that have to be analyzed, these noises are eliminated by means of notch filters. These last filters are needed since there may be

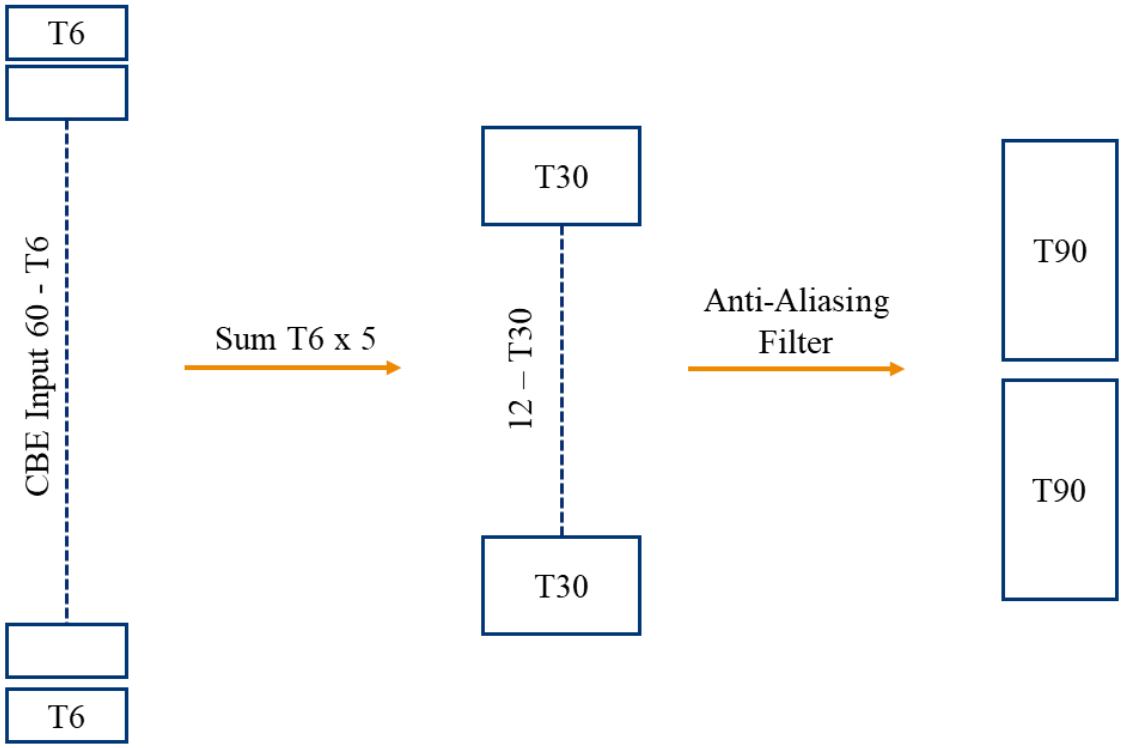


Figure 3.2.6: Sum and Decimation procedure

noises coming by known sources, such as some specific gear groups or driveline configurations which are not related to the unbalance and so have to be eliminated. The frequency of these noises is known by experimental tests, meaning that the notch filter can be designed accordingly.

At the end of the previous procedure the resulting signal is composed by several arrays of two elements for each engine order (A,B,C…). It was noticed that at different speeds or gears, the signal provided by the unbalance estimation procedure is not homogeneous, by looking at Figure 3.2.7 it is clear how after an up-shift (yellow line) the unbalance signal changes its magnitude, in order to ensure the best performances of the following strategies this signal must be multiplied by gains that help to equalize the signal over different speeds and gears Figure 3.2.8.

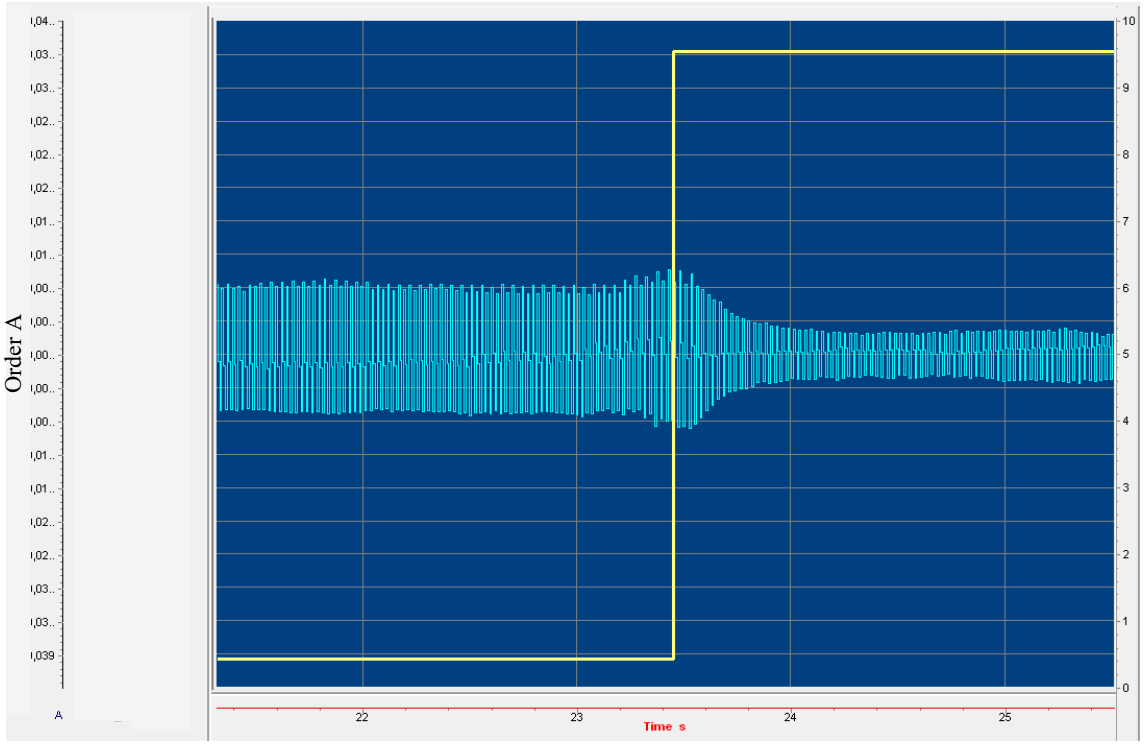


Figure 3.2.7: Order signal (Light Blue) at different Gears (Yellow)

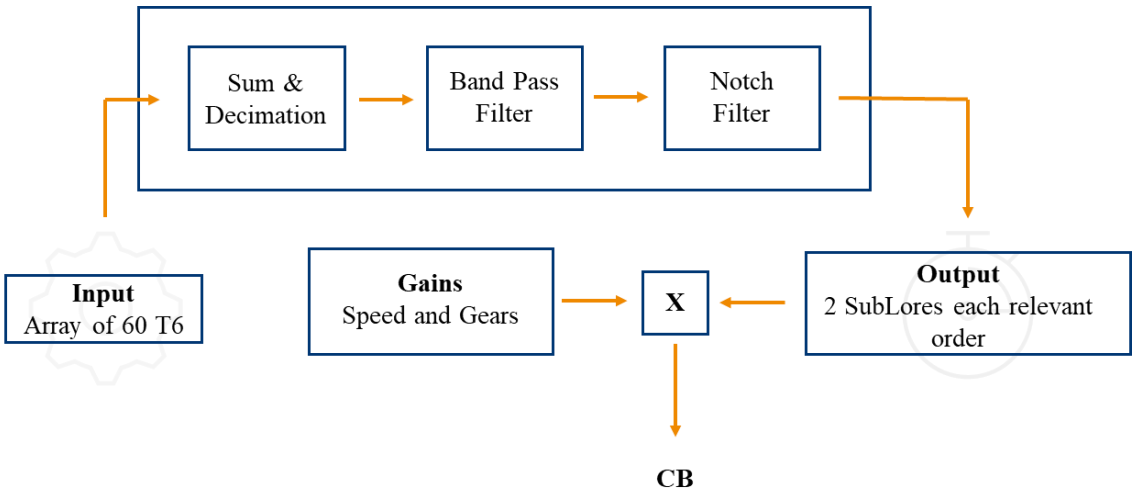


Figure 3.2.8: CBE with Speed and Gears Gains.

3.2.2 Deep look into CBE output

In this section it is explained the nature of the orders signal. The acquisitions that will follow were measured on a 6 cylinders engine, this means that 6 firing events are present in an engine cycle. The following signals are acquired at Lores C meaning the sampling is performed 6 times each engine cycle, therefore it has order 3.

In Figure 3.2.9 it is shown the signal relative to the order A = 0.5 (event occurring once each engine cycle), each step represents a Lores C. It is possible to notice how the signal is periodic each 6 steps which means each engine cycle.The shape is the one of a sinusoid with period 6 Lores C. Instead, by looking at Figure 3.2.10 which is the signal corresponding to order B = 1 the period is half of the previous (3 Lores C) since the signal is relative to events happening twice each engine cycle. Again in Figure 3.2.11, is the signal relative to order C = 1.5 (events happening 3 times each engine cycle and the signal has 1/3 the period of order A (2 Lores C).

The amplitude of the signal determines which is the predominant component, for example, during the test that produced the acquisitions below only one cylinder was drifted, indeed, the signal corresponding to the order A (Events happening once each engine cycle) is the one with the biggest amplitude, notice that the scales through the acquisitions are the same. These signals are later forwarded to the Cylinder Balance control that transforms these information in corrections to the single injector.

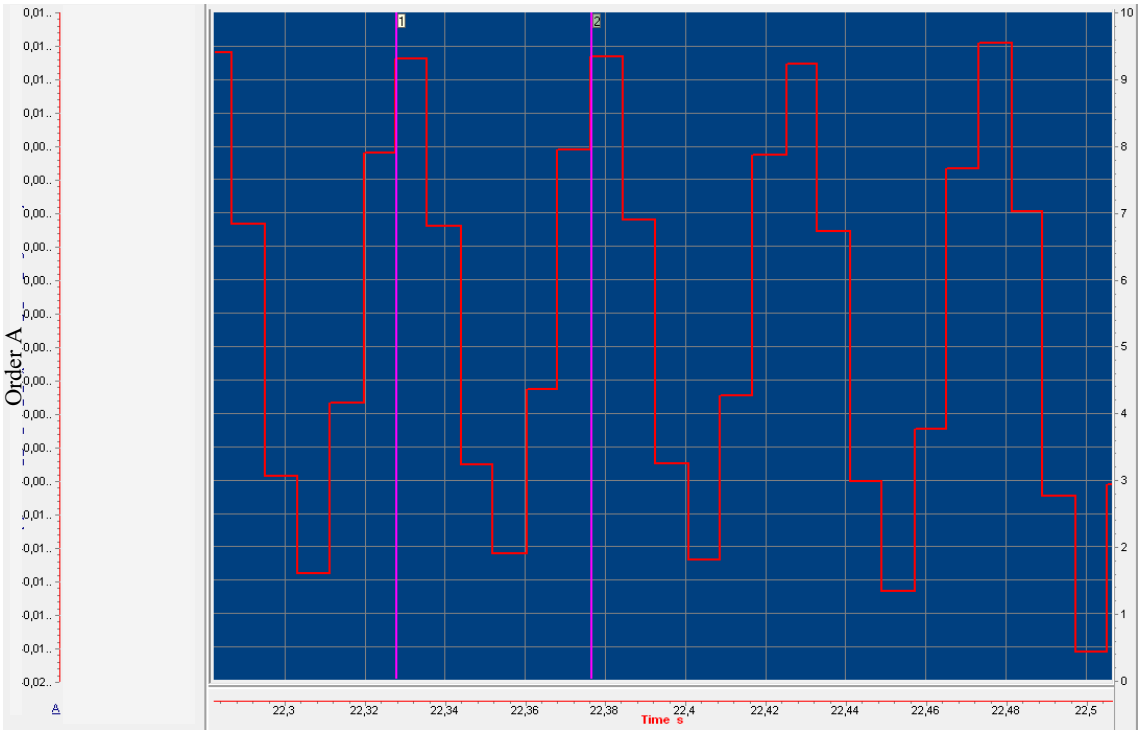


Figure 3.2.9: Order A (0.5) for a 6 cylinders engine.

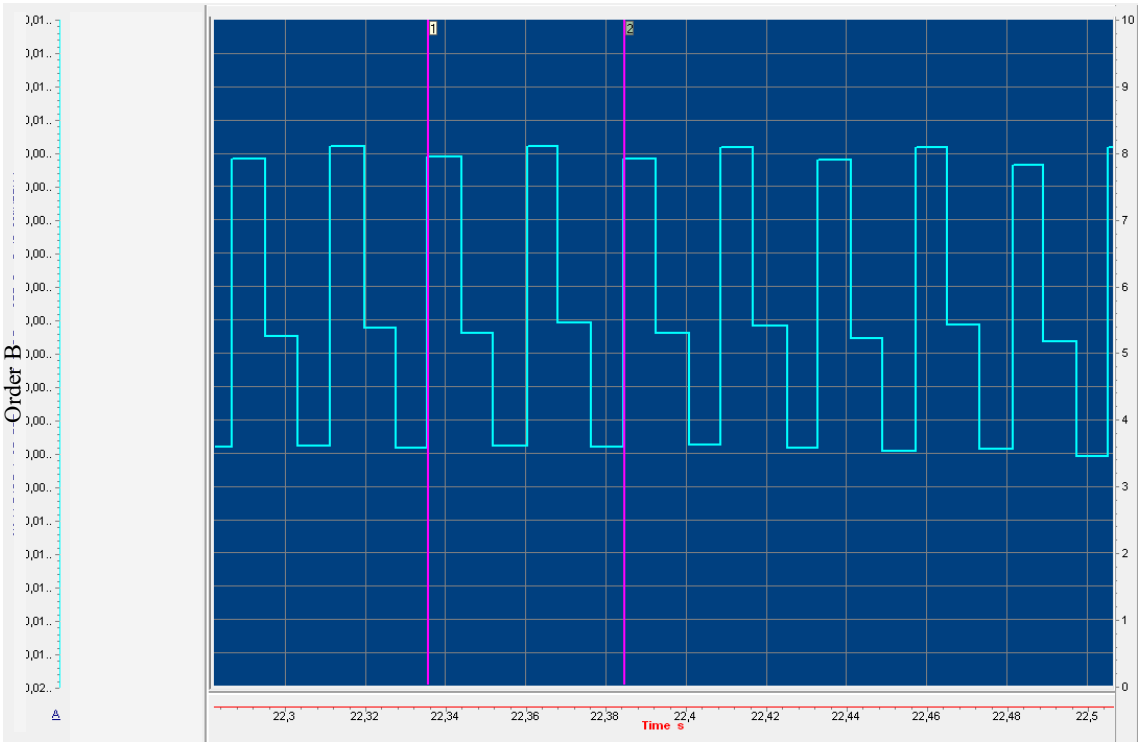


Figure 3.2.10: Order B (1) for 6 cylinders engine

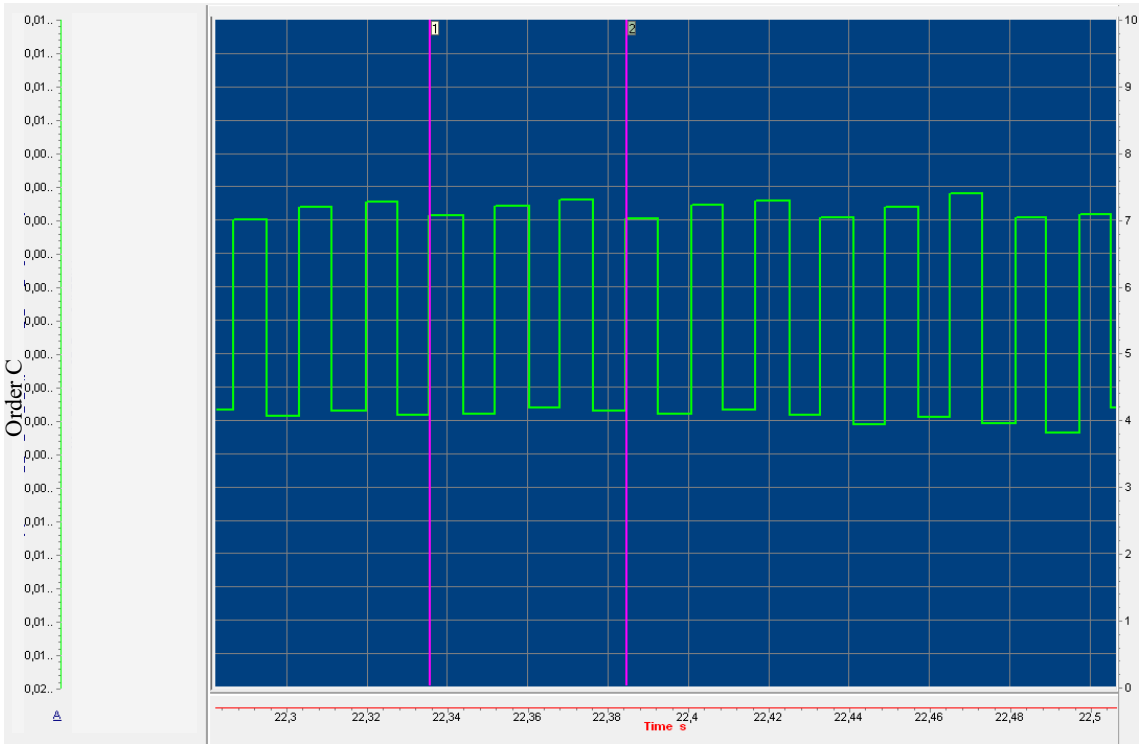


Figure 3.2.11: Unbalance order C (1.5) for 6 cylinders engine

3.3 Unbalance Correction

The information coming from CBE in terms of orders signals has to be converted into corrections to the single injectors in order to reduce the cause of the unbalance. The control system that is responsible for the translation of the order signals into correction is the so called Cylinder Balance CB Figure 3.3.1.

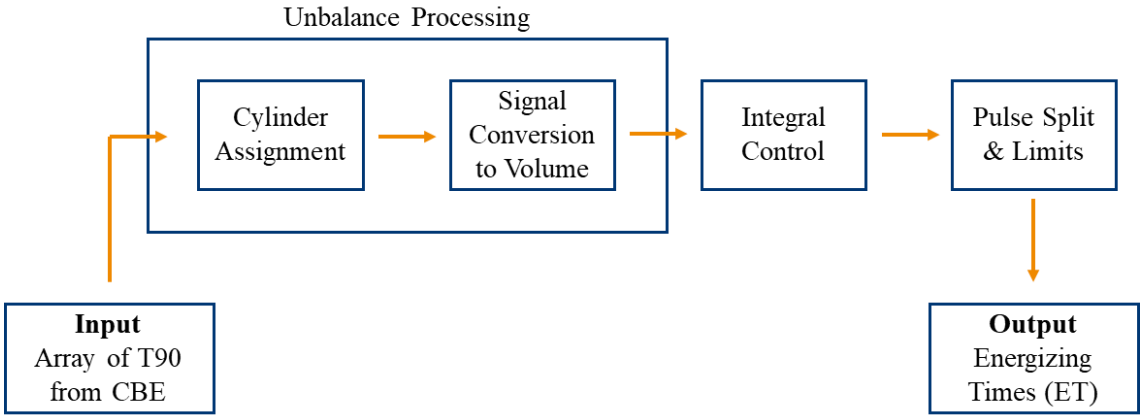


Figure 3.3.1: Cylinder Balancing Flow Chart

3.3.1 Cylinder Assignment

As already said, CBE provides arrays with two Sub Lores for each relevant order. In order for the CB to understand which cylinder is related to a Sub Lores and consequently which is the unbalanced one, each of the Sub Lores have to be assigned to the right cylinder. This action is performed through the Cylinder Assignment.

Taking into account a 4 cylinders engine, the CBE provides the CB with an array of 8 Sub Lores (T90 in this case), in order to assign the right T90 to the right cylinder there is a calibratable Map that allows to perform the assignment. The principle of working of this Map is schematized in Figure 3.3.2.

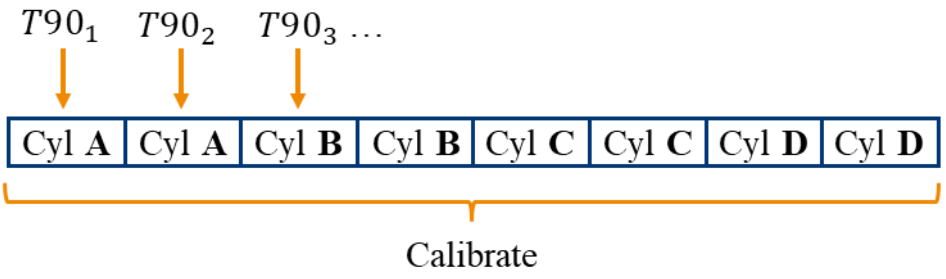


Figure 3.3.2: Cylinder Assignment

Considering an engine with a firing order of the type A-B-C-D the most logical way to perform an assignment is as the one in Figure 3.3.2. However, due to ECU's computational delays, noises etc. this type of assignment may not always be the best performing, for example, during testing it was noticed that at different loads and speeds the best performing assignment type is not always the same. It was then decided to increase the size of the assignment map to get a more flexible one with different types of assignments Figure 3.3.3. In this way the calibrator can decide between three different assignment types that should be enough to cover all situations.

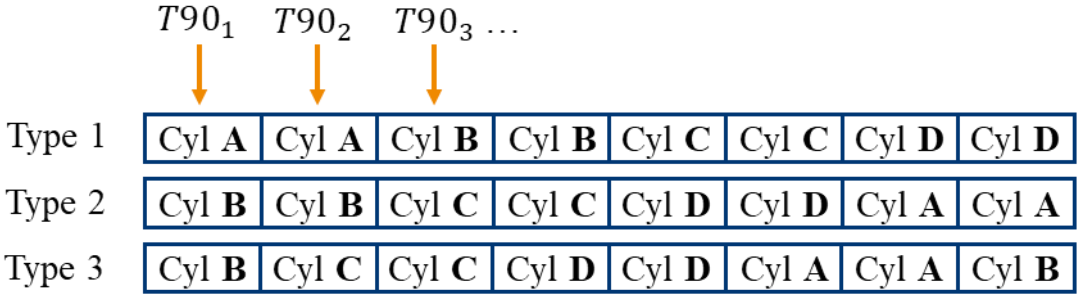


Figure 3.3.3: Cylinder Assignment complete Map

During testing, it was noticed that the assignment type that performs the best, strongly depends on the engine point (Speed & Fuel Request). In order to use a different type of assignment depending on the engine point, a second map is introduced. This new map determines what type of assignment has to be used (i.e. which line of the assignment type in Figure 3.3.3), depending on the engine point Figure 3.3.4.

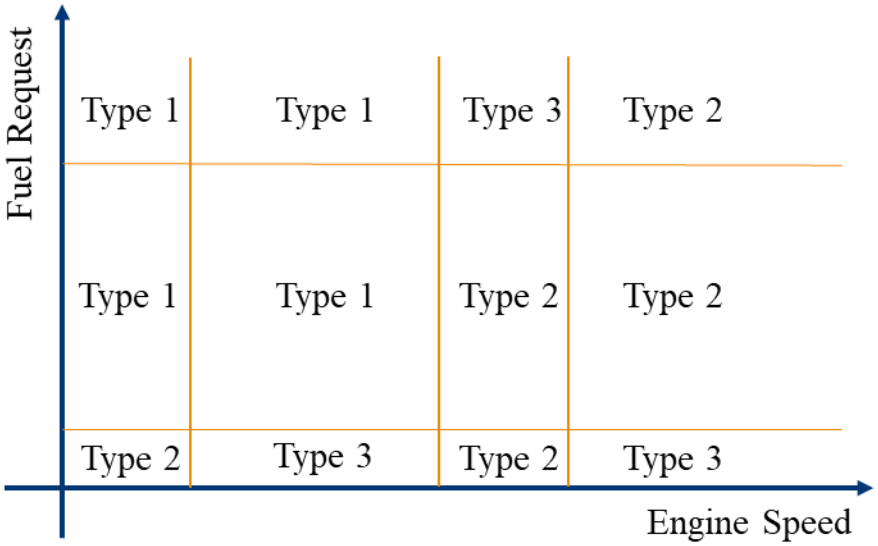


Figure 3.3.4: Example of Assignment Type selection Map

This last map has to be calibrated by performing many tests at the Dyno, possibly covering all the points in the working range of the engine. There are two things that have to be calibrated for this map:

1) **Breakpoints**: They are the thresholds that divide different working areas in the map, in Figure 3.3.4 are represented by the orange lines. These thresholds are provided with hysteresis in order to avoid toggling between two different assignment types.

2) Best performing **Assignment Type** within the working areas that were previously determined through the breakpoints calibration. During this calibration it has to be evaluated which assignment type is the best performing at a given engine point.

3.3.2 Signal Conversion to Volume

Two Sub Lores C are contained in one stroke, once the expansion stroke is measured for one cylinder, it will be measured again after a number of Sub Lores C determined by the number of cylinders. For example, in a 6 cylinders engine when the two Sub Lores of cylinder A are measured they will be measured again after 10 Sub Lores. During this period of time the information coming from CBE has to be frozen to the time at which it was measured.

In Figure 3.3.5 it is shown an acquisition on a 6 cylinders engine, the order 0.5 signal (Red line) together with the unbalance signal of cylinder A (Green line) are shown and acquired at Lores C. The second and first line are respectively father and son, since the unbalance signal relative to one cylinder, is nothing more than the order signal at one Lores C (Previously assigned) and kept constant for 6 Lores C (One Engine Cycle).

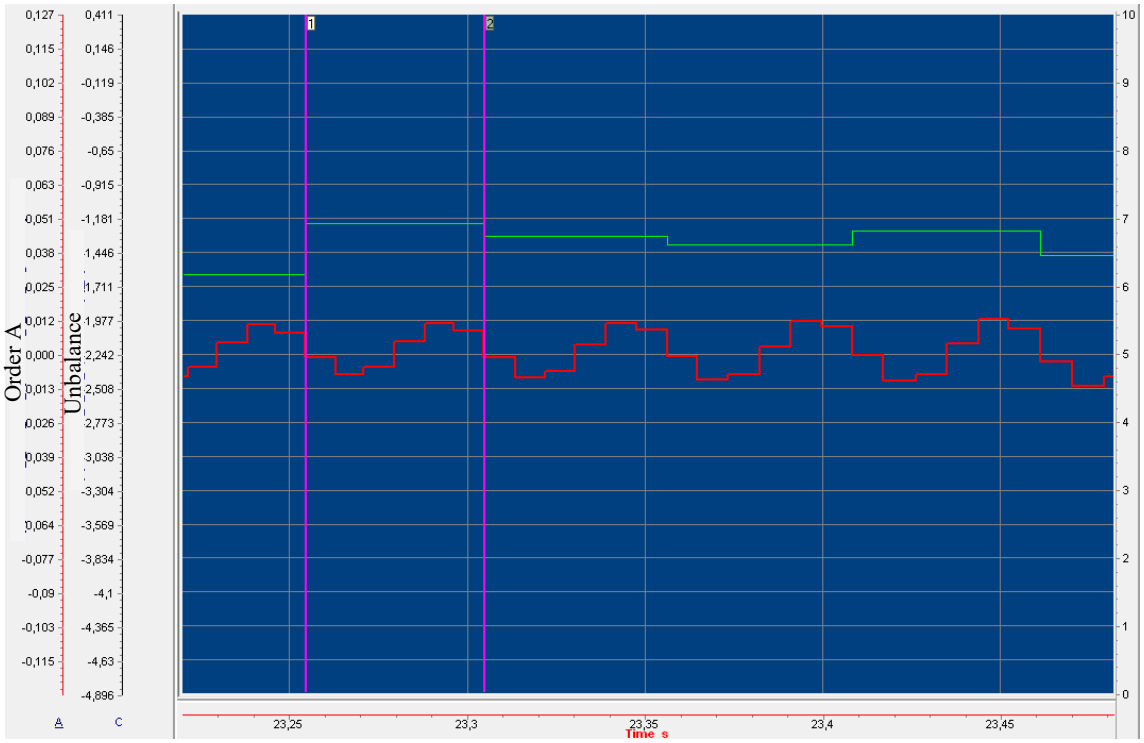


Figure 3.3.5: Order 0.5 Signal (Red Line) and Unbalance Signal for cylinder A (Green Line)

If the other cylinder's unbalance signals are added to Figure 3.3.5 they are shifted one Lores C one to another, since they are assigned one Lores C later, Figure 3.3.6.

In order to pass from the Order signal (Time) to the Unbalance signal (Quantity), the first one, that was previously assigned, is multiplied by some gains that have to be calibrated during testing.

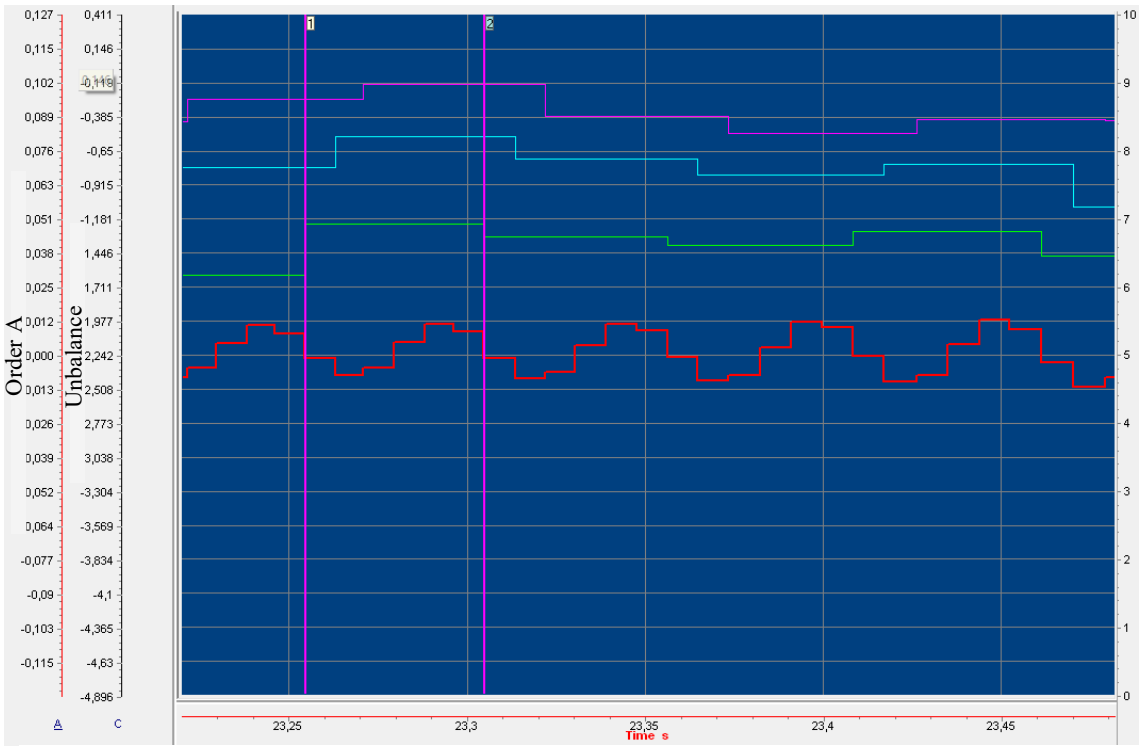


Figure 3.3.6: Order 0.5 Signal (Red Line) and Unbalance signal for Cylinders A, B, C (Respectively Green, light blue and Blue Lines)

In Figure 3.3.7 the unbalance signals for all cylinders are shown, the red line is the signal relative to the positively drifted cylinder (Cylinder A). This signal is the lowest negative, one may intuitively think that since the drift is positive the unbalance signal should be positive too, however, a positive drift should produce a negative correction, in these terms the unbalance signal is more coherent.

Once the order signal is converted in unbalance signal, which is measured in mm^3, it can be fed to the integral control which will take care of reducing its entity by delivering corrections to the injectors.

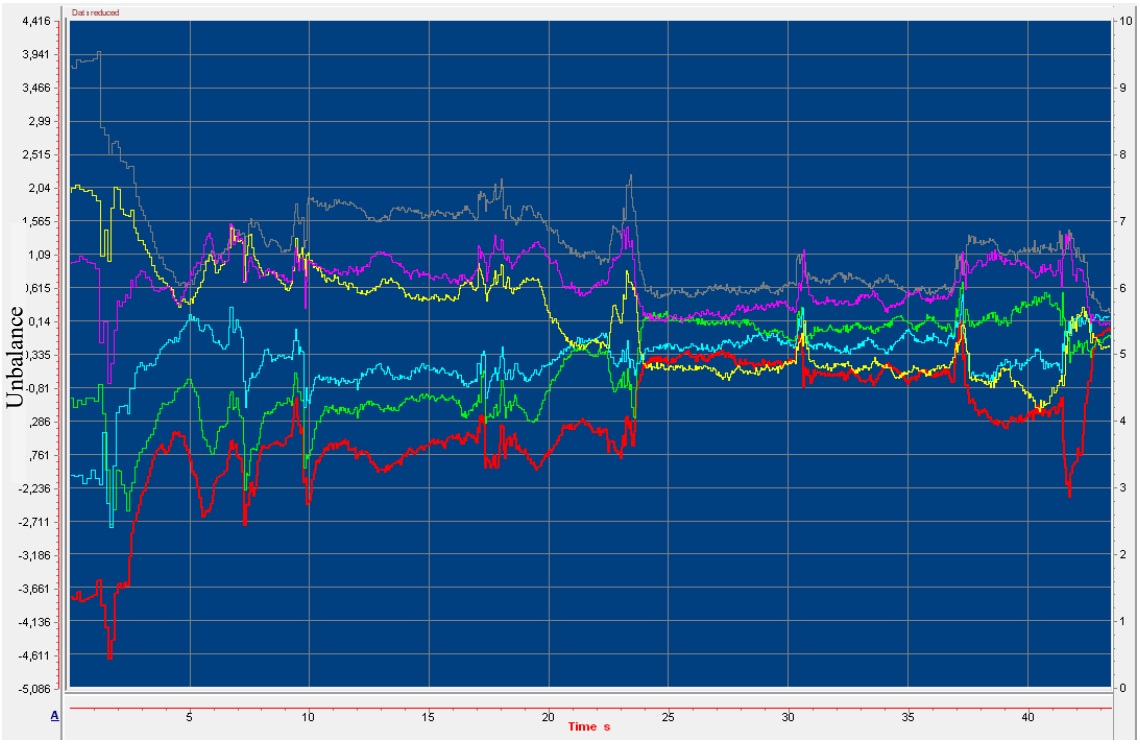


Figure 3.3.7: Unbalances of all 6 cylinders

3.3.3 Integral Control

The Unbalance signal that comes out from the previous steps is the input signal of the real control. This control is purely integral, it takes the Unbalance signal as Input and outputs the corrections that have to be delivered to the injectors. These corrections are quantity corrections $[mm^3]$. For some injectors, corrections can be delivered in terms of Δmm^3 for other injectors this information must be converted into Energizing Time (ET) μs, this conversion will be explained in the next chapter.

Getting back to the control, a simple schematic representing the principle of working of the closed loop control is represented in Figure 3.3.8. It is represented the control on the

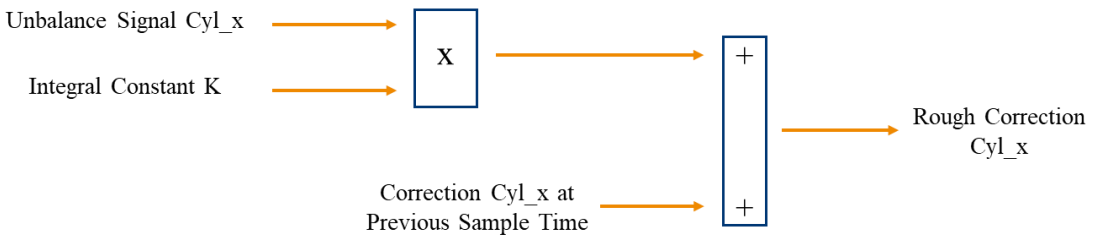


Figure 3.3.8: Integral control on single cylinder

single cylinder, the unbalance signal relative to the analyzed cylinder is at first multiplied by the integral constant that has to be calibrated, to the resulting value it is added the correction at the previous step. The last operation determines why it is an integral control.

One important aspect has to be considered now, the fuel request of the engine is the total amount of fuel that is needed by all the cylinders in the engine at a given time instant, it is simply calculated as the sum of the single fuel requests of the cylinders.

When the fuel request is set to a value, for example 20 mm^3, after the CB control has calculated the corrections for the single cylinders the total fuel request must still remain the same. It is paramount important since the fuel request that was set by the engine management system takes into account emissions, fuel consumption etc.

In order to ensure that this conditions are met, a zero average regulator must be added to the control system Figure 3.3.9. This sums the corrections for all the cylinders, divides them by the number of cylinders and subtract this result from the single corrections. In the end the sum of all corrections will result in 0, meaning that the total fuel request will remain the one set by the engine management system.

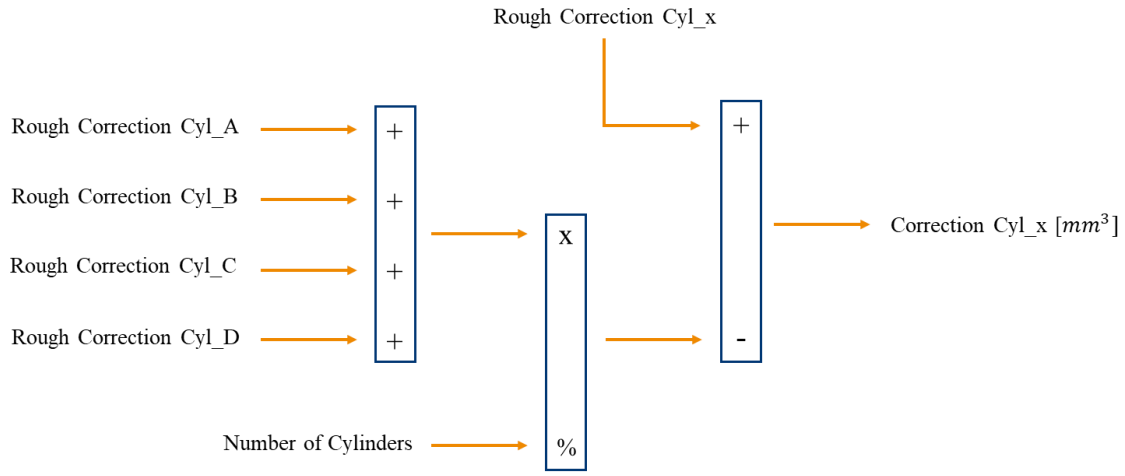


Figure 3.3.9: Average removal from corrections

The only degree of freedom of the integral control is represented by the integral constant K, for the CB control system it is not unique but can assume different values depending on the engine operating point.

The integral constant is picked by the control from a Map that needs to be calibrated by means of many tests at the Dyno. This Map is 6x8 in the current control, it has 6 speeds break points and 8 fuel request break points. Each constant is calibrated by imposing a drift on a cylinder and by enabling the control, the unbalance response is evaluated in terms of convergence time and overshoot.

The CB provides two calibratable maps, one for stationary condition and a second one for transient condition, a transient situation is determined by calibrating some thresholds on $\Delta Fuel\ Request$ and $\Delta Engine\ Speed$. When a transient is detected the integral constants map that is used is a dedicated one calibrated in transient conditions.

Having two dedicated maps allows to be more gentle with the integral constant during steady state operation, allowing the unbalance signal to avoid overshoots while converging. In transient the integral constant value can be increased in order to allow the unbalance signal to react faster to changes and being able to catch up.

An example of test that has to be performed in order to calibrate these maps is shown in Figure 3.3.10 and Figure 3.3.11. These tests are at stationary conditions, engine speed is set to 1500 rpm and fuel request is blocked to 30 mm^3, cylinder A (Red Line) is negatively drifted (Positive unbalance signal), once the unbalance signals are stabilized the CB control is enabled (Thick yellow line).

The test in Figure 3.3.10 is performed with an integral constant equal to 0.3 and once the CB control is enabled the unbalances response have really bad oscillations, meaning that the integral constant is too big and needs to be decreased. The second test Figure 3.3.11 is performed with an integral constant equal to 0.1, it can be noticed how the response is now acceptable for the purpose.

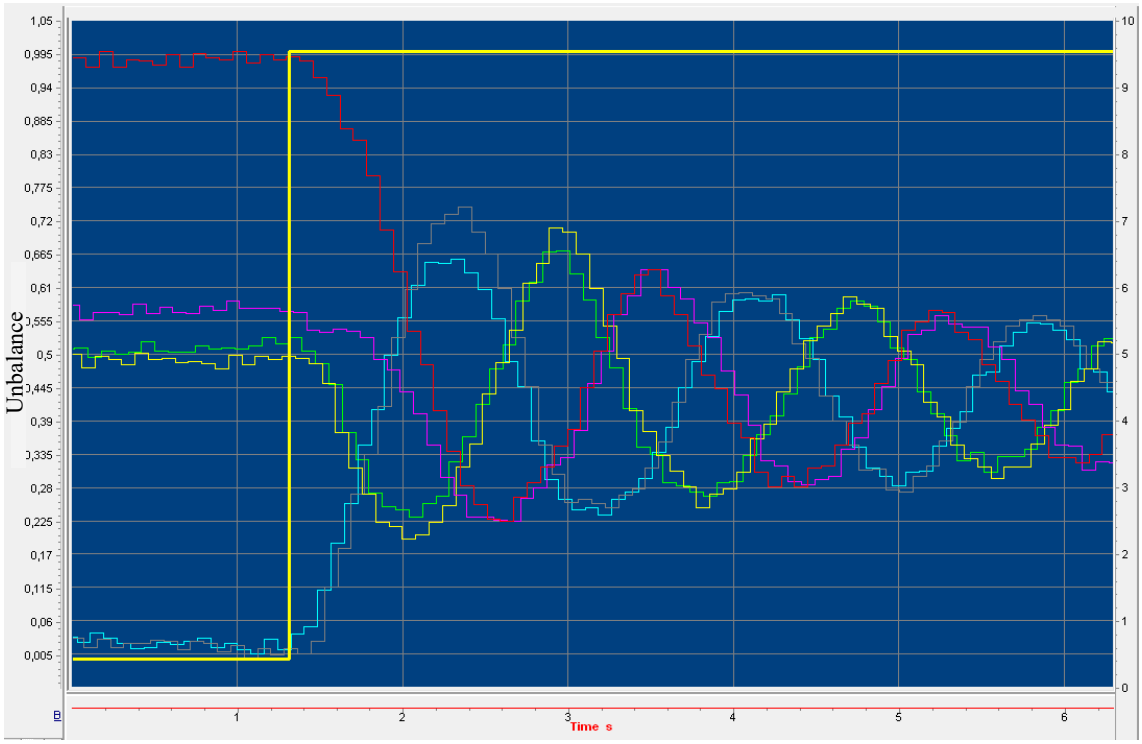


Figure 3.3.10: Unbalances response after CB enabling, integral constant K = 0.3

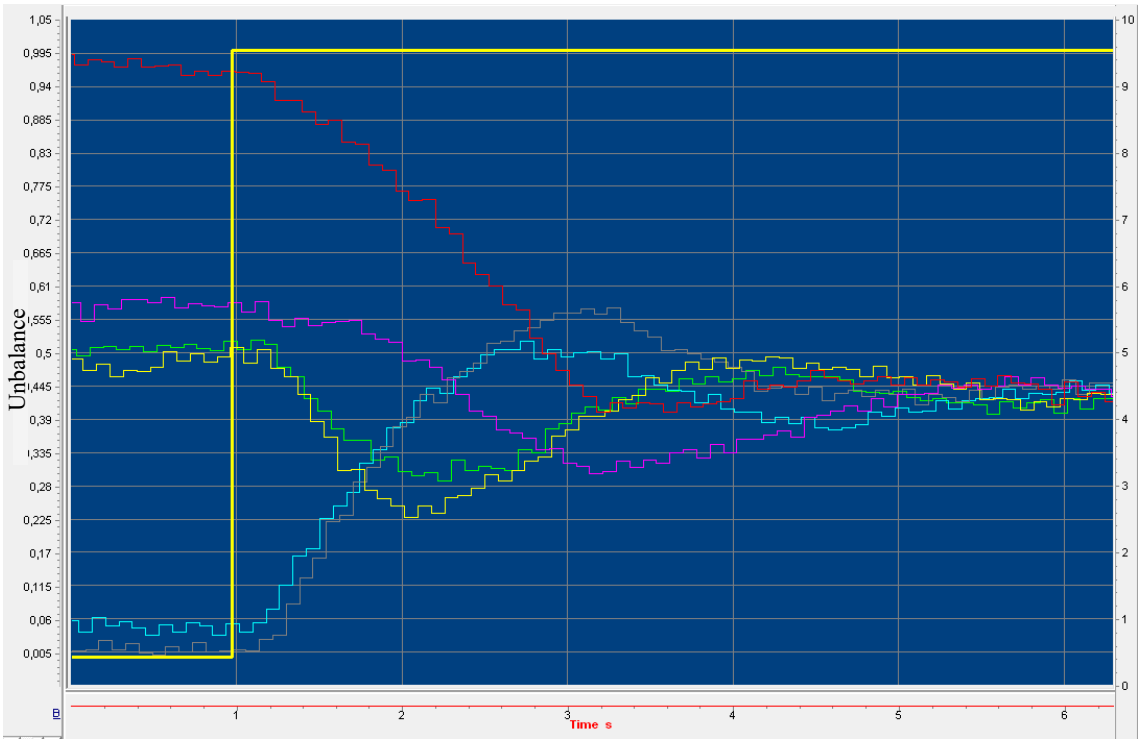


Figure 3.3.11: Unbalances response after CB enabling, integral constant K = 0.1

This is a kind of test that needs to be performed for each breakpoint of the map, meaning that for a 6x8 map at least 48 tests need to be performed. In the test highlighted above for examples 2 tests were needed, it is now clear how time and test demanding this calibration is.

3.3.4 Pulse Split and Limits

As highlighted in section 2.4.1, fuel injection is not performed in one step but in multiple injection events. The number of pulses during fuel injection varies depending on the injection pattern. The types of pulses may be:

- Pre Injections (R): Injection events that happen before the main Injection, it has torque smoothening purposes.

- Main Injection (M): Is the one that is mainly responsible for torque production.

- After Injection (A): Come after the main Injection, usually inserted for pollutant reduction purposes.

An example of injection pattern could be R2 – R1 – M – A, notice that the only pulse that is 100% torque forming is the Main, all the other pulses have a torque forming percentage lower than 100. This is because part of the fuel is utilized for secondary purposes.

All of the pulses have an upper saturation limit, this is imposed in order to avoid any engine damage, also, a lower limit must be set, this instead is needed because if the fuel quantity is too low the engine misfires (Figure 3.3.12).

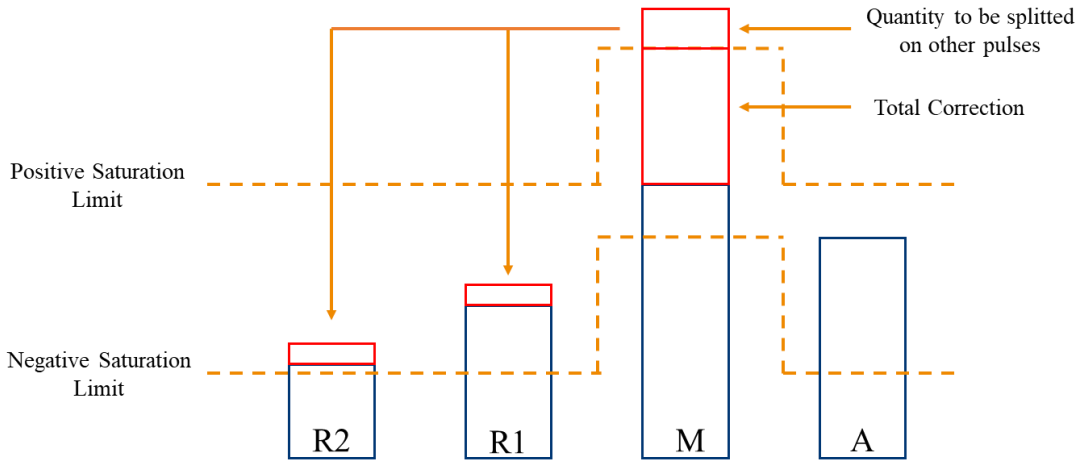


Figure 3.3.12: Pulse Split and Limits

These aspects have to be taken into account when delivering corrections to the injectors, for example, imagine having a main pulse of 10 mm^3, a total correction of 3.5 mm^3 and an upper saturation limit on the main pulse of 12 mm^3.

When a similar situation occurs, the CB control decides to split the remaining quantity that cannot be injected with the main pulse. Getting back to the previous example, there are 1.5 mm^3 that can not be injected with the main pulse, they are then injected with the first of the pre injection pulses and if the same situation occurs the procedure is repeated until there are available pulses. When all the pulses reach their saturation limits the CB is said to be saturated and has lost its torque compensation capability. Same reasoning can be applied to the negative correction.

The split quantities are ruled by a mathematical law that takes into account the torque forming fuel quantity for each pulse, the number of pulses and the total correction in the following way.

- $\boldsymbol{CB_{QDelta}}$: is the exceeding CB correction quantity.
- $\boldsymbol{Fuel\_Desired\_X_{Pulse\ Torque\ Forming}}$: is the basic quantity (i.e. the fuel request set by the ECU) of the pilot pulses or the torque forming part of the after pulses.
- $\boldsymbol{\sum Fuel_{EnablePulseTorqueForming}}$: is the current torque forming quantity of all enabled pulses.

- $\boldsymbol{Fuel_{MainTorqueForming}}$: is the current main pulse torque forming quantity.

$$Corr_{Pulse.x} = CB_{QDelta} \frac{Fuel\_Desired\_X_{Pulse\ Torque\ Forming}}{\sum Fuel_{EnablePulseTorqueForming} - Fuel_{MainTorqueForming}}$$

In this way the total correction is split taking into account which impulse has "more room" for corrections.

It is very important to avoid saturation of the CB otherwise the control is not able to compensate anymore. This happens because when the CB is saturated, even if the value of the corrections keeps increasing, these cannot actually be injected. In this situation the system (i.e. the engine) is not corrected and it keeps the unbalances, the corrections continue increasing since the system does not detect a convergence of the unbalances and corrections diverge.

3.3.5 Correction conversion from Δmm^3 to ΔET

As explained in the previous section, the CB control calculates the corrections for each cylinder in terms of fuel quantity Δmm^3. Some injectors can manage the corrections exploited in these terms but others cannot.

In the engine analyzed during testing the kind of used injectors are of those that need to have the corrections provided in energizing times. In this case the corrections provided by the CB control need to be translated into energizing times. In order to achieve so a new map is built starting from the injector map and it makes the translation possible.

The injector map is as the one in Table 1, in order to pick the right energizing time for a injector in a given engine point one has to enter the map with the rail pressure and the fuel quantity requested from the engine management system. The values in the map are then expressed as $[\mu s]$.

Table 1: Injector Table (Partial)

mm^3/MPa	12,5	15	20	25	30	40	50
0	0	0	0	0	0	0	0
0,203	355	310	238	214	206	184	170
1	468	410	335	295	270	227	210
1,5	562	483	384	334	305	264	230
2	646	538	436	370	336	289	262
2,5	718	602	479	411	367	312	281
3	788	658	517	446	400	336	301
4	904	753	594	502	455	381	334
5	1004	830	661	563	500	424	370
6	1094	907	723	617	549	457	404

The Map that exploits the translation from Δmm^3 to ΔET is retrieved from the injector table by performing the following calculation for each fuel breakpoint at constant pressure breakpoint.

$$K_{conversion}(1mm^3@20Mpa) = \frac{ET(1.5mm^3@20MPa) - ET(1mm^3@20MPa)}{1.5mm^3 - 1mm^3} \qquad (1)$$

The resulting map is then Table 2, each value in the map is then expressed as $\left[\frac{\mu s}{mm^3}\right]$

Table 2: Conversion Gains

mm^3/MPa	12,5	15	20	25	30	40	50
0,203125	302,7	274,8	259,0	240,5	224,6	196,9	178,1
1	274,2	243,2	219,9	203,0	191,5	169,6	151,6
1,5	233,2	204,4	182,6	166,6	156,2	134,6	122,7
2	188,9	163,7	139,9	127,2	115,9	98,9	86,9
2,5	173,1	148,0	126,1	113,3	104,6	88,8	78,3
3	154,8	131,4	112,7	99,7	91,3	77,2	67,7
4	132,1	110,1	92,8	82,8	74,3	62,1	54,2
5	121,2	100,6	83,4	73,4	66,8	54,7	47,8
6	113,5	93,4	76,6	66,9	60,6	54,2	42,6

In this way, when corrections are calculated by the CB control in terms of mm^3, by multiplying them with their correspondent gain the correction is translated in μs.

3.4 Dead Band

The CB control aim is to reduce as much as possible the spread between the unbalances of the cylinders. When Noise Vibration and Harshness (NVH) requirements are met, CB corrections should no longer increase.

In order to exploit this logic, two thresholds on the unbalance signals must be inserted. These two, set the minimum negative and the maximum positive value of the unbalance signal for which the CB correction on the cylinder under consideration must be frozen. Under these conditions the unbalance signal is set to zero and the corrections no longer vary.

NVH requirements are evaluated taking into account the Indicated Mean Effective Pressure IMEP. The IMEP signal is the average pressure, that is induced in the combustion chamber during the complete thermodynamic cycle in internal combustion engines. In multi cylinders engines the IMEP signal may vary in between cylinders, the NVH requirements are met if the spread between the IMEP of all cylinders lay in a determined range.

In order to calibrate the two unbalance thresholds for the dead band, the IMEP signal becomes fundamental. Indeed, the two thresholds are determined as the value of unbalance for which the IMEP spread between all cylinders is lower than a certain value.

- σ_{IMEP}: $IMEP\ Dispersion$
- σ_{Limit}: $Maximum\ dispersion\ allowed, depending\ on\ the\ application$

$$\sigma_{IMEP} < \sigma_{Limit}$$

3.5 Cranking

During the first engine strokes, after a cranking event, there are some factors that can degrade the Cylinder Balancing behavior. This may happen due to the first combustions and to the sudden change in fuel request and crankshaft speed. Also, at the engine start up some values in the ECU must be restored, since they are lost when the engine is turned off and the key is extracted, this may induce the CB to provide erroneous corrections. These factors created the need to delay the CB control enabling when a cranking event occurs.

A Boolean flag is utilized whenever a cranking event is detected, when this flag is set to true the CB strategy is turned off. When the flag is set to false a certain number of samples

are waited before the activation of the CB. The number of samples that have to be waited has to be calibrated.

4 CB Control system calibration and Results

In Chapter 3 it was explained how the Cylinder Balance control works and its main features. Once the control strategy is defined it has to be adapted to the specific application, it means that depending on the type of engine that is under consideration, the CB Control System has to be calibrated in order to perform in the best possible way on the specific application.

The presented control system has many degrees of freedom which will be referred to as calibration labels. By calibrating these labels one can reach the best performances on different engines, in the following chapters will be listed the calibration procedures for the labels for which it was highlighted an area of improvement in the calibration time effort. The type of engine on which the CB is calibrated during testing has the following specifications:

DURAMAX LZ0

- 3.0 Liters
- 277 hp
- 620 Nm
- 6 Cylinders
- 2 Wheels Drive
- 12 Gears Automatic Transmission

The tests were performed at the Dyno test bench (Figure 4.0.0). The reason why the tests are performed on vehicle, instead of on an Engine test bench, is that the performances of the CB control are highly influenced by the driveline behavior. Indeed, in the previous chapters it was explained how some calibrations are depending on the driveline configurations (i.e. driveline gains in the Unbalance Estimation, Chapter 3.2).

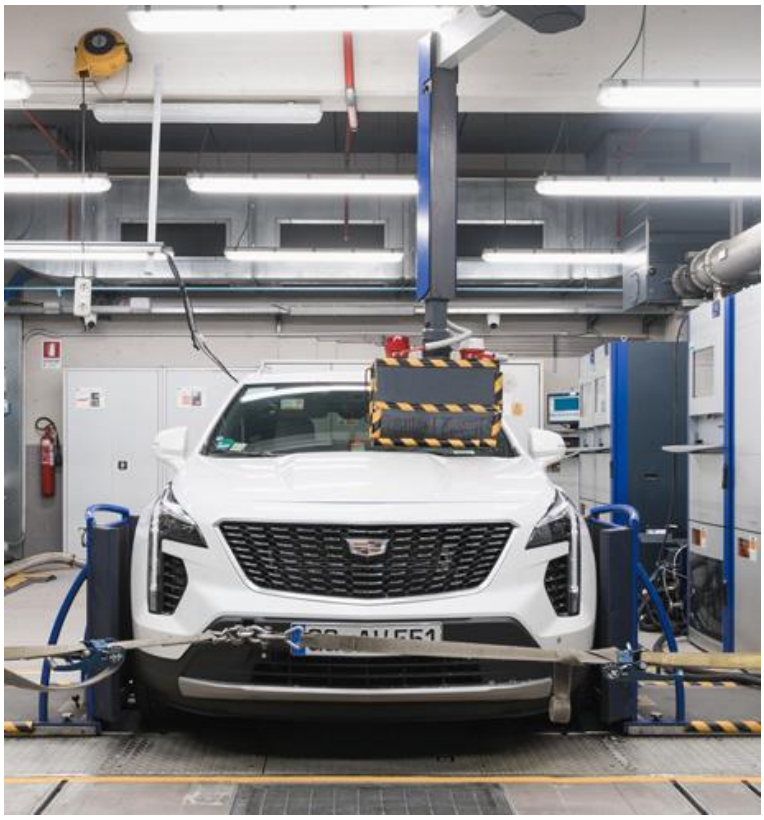


Figure 4.0.0: Roller Test Bench.

4.1 CBE Calibration

At the top of the cylinder balancing control there is the cylinder balance estimator, it provides a signal for each order whose magnitude depends on the types of drifts present in the engine. Calibrating this subsystem is paramount important in order to avoid the CB to provide misleading corrections. One of the most influent calibrations in the CBE system is the mean value removal from the Lores samples.

Imagine having a sudden increase in engine speed, which is set through the accelerator pedal, in this situation the crankshaft increases its speed resulting in a sequence of Lores samples that becomes smaller and smaller while the speed increases (Figure 3.2.2). This situation would be interpreted by the CB as a sequence of cylinders that are positively drifted (i.e. positive drift means that more quantity is injected, resulting in greater pressure in the combustion chamber, greater torque at the crankshaft and in the end increasing engine speed), instead, it is just a normal acceleration of the engine. Moreover in situations such as a gearshift in which the clutch is disengaged and suddenly engaged again there is a spike of engine speed that induces a strong transient and can be misinterpreted.

In order to mitigate this effect the CBE is provided with two calibration labels whose job is to calculate the average value over a number n of Sub-Lores samples and remove this

result from a specific sample at each sampling time, in an ideal situation by applying this procedure to Figure 3.2.2 (i.e. removing the average speed) Figure 3.2.1 is obtained. A schematic representing how this calibration works is shown in Figure 4.1.1.

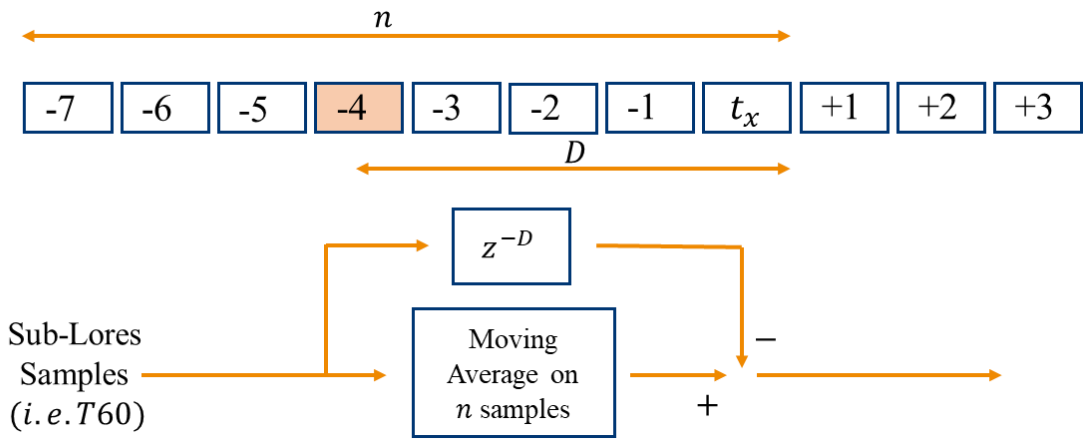


Figure 4.1.1: CBE Mean Value Removal

The mean value of the Sub-Lores samples is calculated over a number n of them that can be decided. The engine under consideration is a 6 cylinder engine, meaning that the sample time in such an application is a T60, in order to average over a crankshaft revolution it was decided to try a number $n = 6$ and removing the average to the T60 in the middle by setting D = 3.

To evaluate the performances of this filtering it was performed a FTP72 test, with a drifted cylinder, that was later compared to another FTP72 test in which the mean removal functionality is not implemented (Figure 4.1.2). In order to better understand the differences between the unbalances signals in the two cases it is helpful to compare the two in the highlighted area in Figure 4.1.2 which is a situation of strong transient (i.e. Gearshifts, increasing/decreasing speed and fuel request). The comparison is shown in Figure 4.1.3, with this kind of parameters for the calibration, the unbalances signals are made even more noisy than the case in which there is no such a feature activated. Indeed, by analyzing the situation when the gearshift occurs a strong noise is induced.

In a second test the window of samples on which the Sub-Lores are averaged is increased to 12 (i.e. $n = 12$ $D = 6$) and the same test was performed and compared to the baseline which is represented by the FTP with no calibration (Figure 4.1.4). In this case when a gearshift occurs the unbalance signal is way smoother than the case with no removal activated, this is paramount important in order to have CB corrections as smooth as possible. It will be noticed later that this feature also helps the assignment to work correctly.

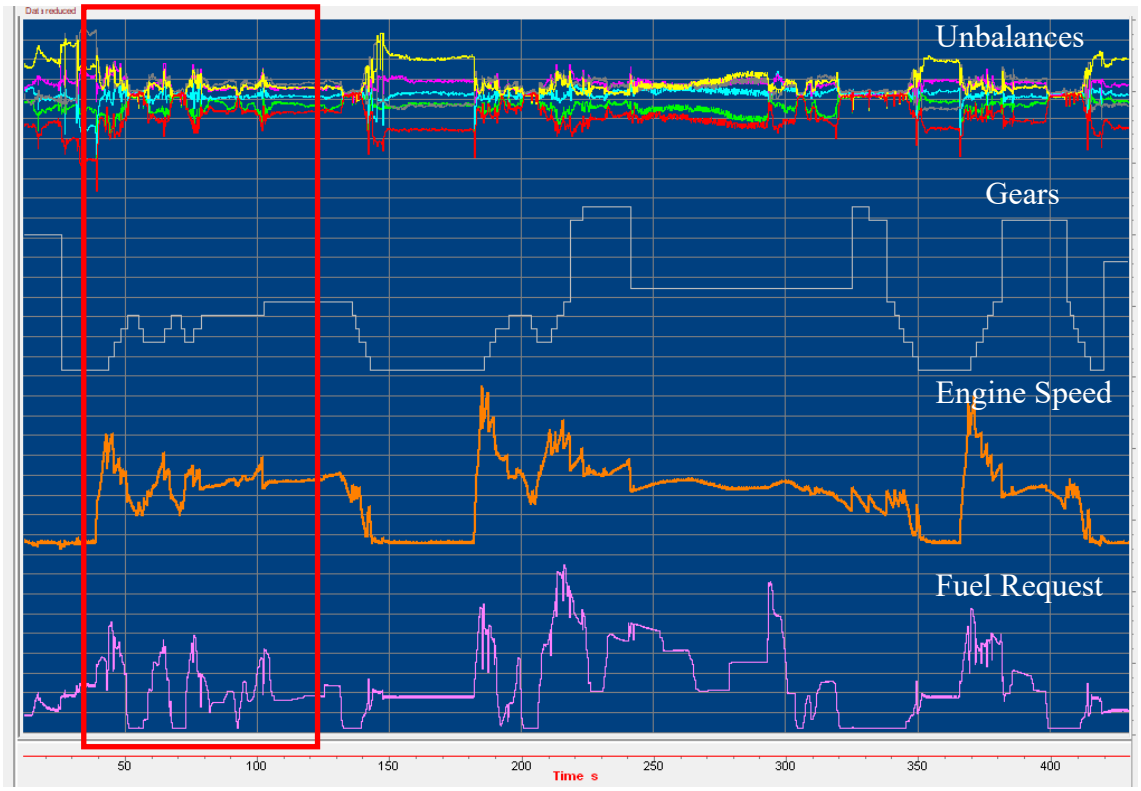


Figure 4.1.2:FTP72 With no Mean Removal

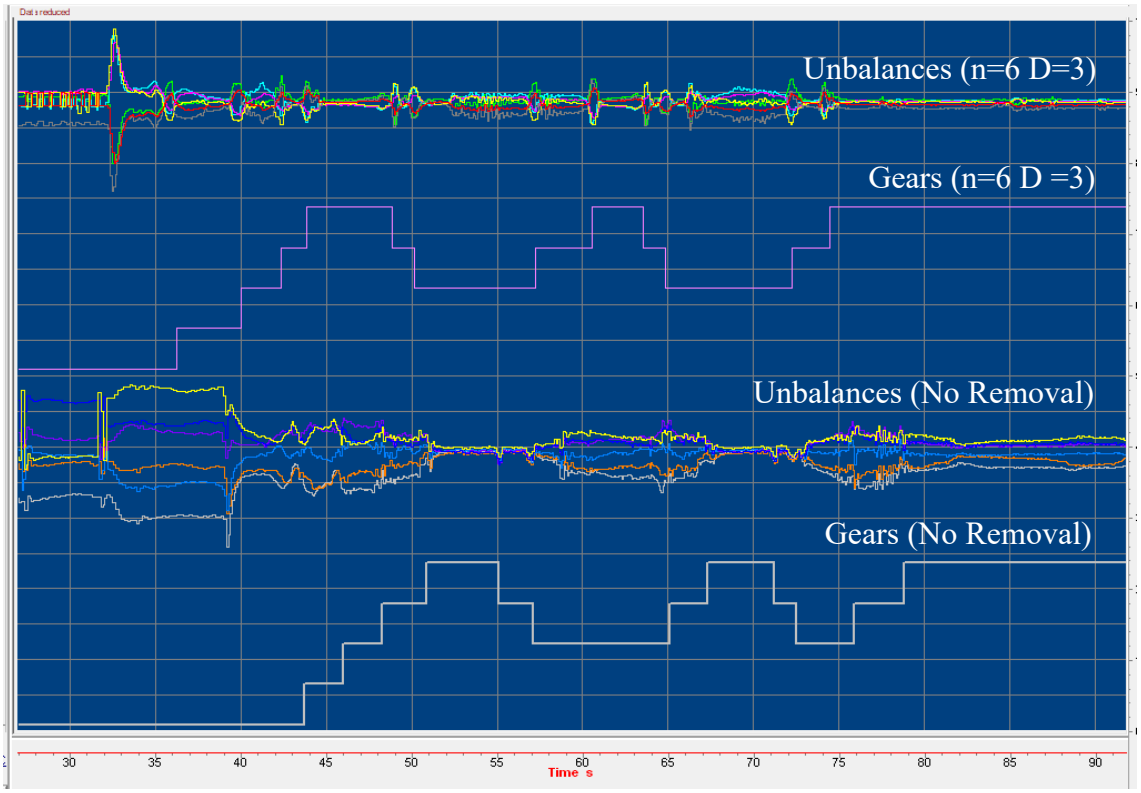


Figure 4.1.3: Mean value removal (n=6 D = 3) compared to the case with no removal

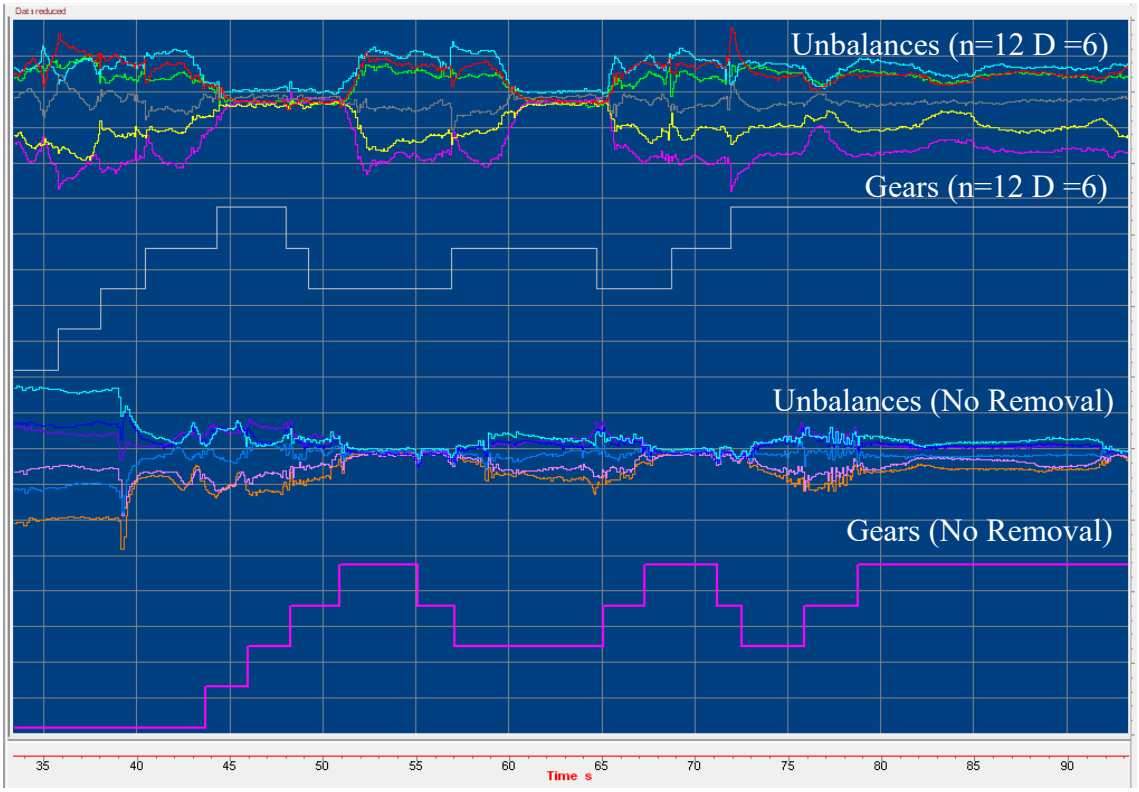


Figure 4.1.4: Mean value removal (n=12 D = 6) compared to the case with no removal.

A tool was developed for the calibration of the cylinder assignment, this allows to evaluate where in a cycle the assignment is not correct. This tool was also adopted to evaluate the effects of the calibration of the mean removal. Most of the times the assignment fails in situations of strong transients (i.e. gearshifts, steep increase of fuel request & speed) because the unbalances signals become very noisy, the mean removal accounts for the smoothening of these signals and as it will be seen its effect is beneficial

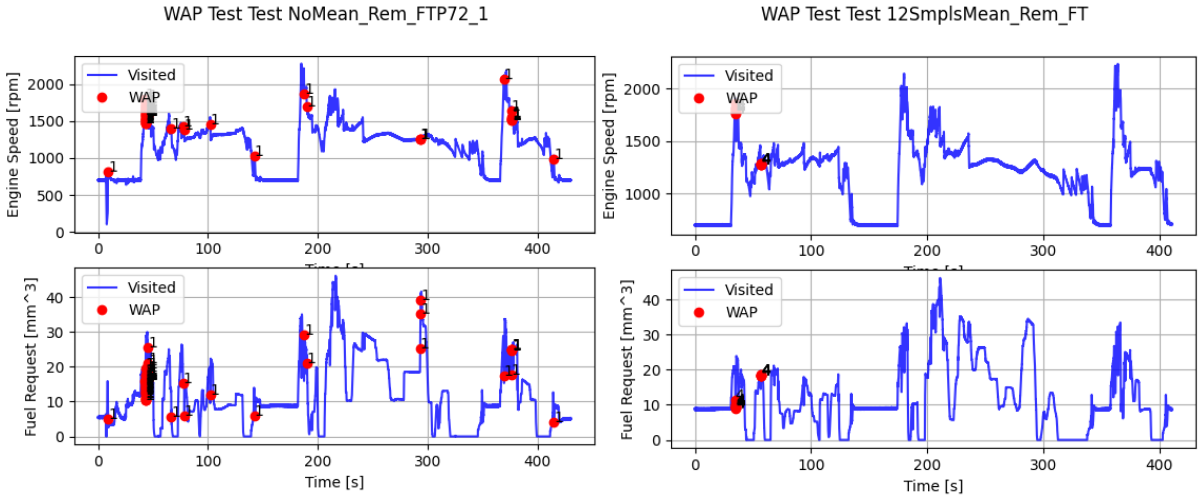


Figure 4.1.5: Effect of Mean Removal on Cylinder Assignment.

for the assignment. In Figure 4.1.5 two FTP72 cycle are shown, one of them is without mean removal enabled the second one is with this feature enabled. The red dots instead is where the assignment has failed, as it can be seen, when the mean removal is not enabled there are way more points in which the assignment fails.

4.2 Enabling Conditions

The CB control can be activated and deactivated under certain conditions (Enabling Conditions) which are of various type, for example, fuel request and engine speed conditions, type of gear engaged, selected combustion mode and so on. In the following section it will be explained which are the conditions under which the CB is enabled and how they are calibrated, taking into account that for the engine model under consideration the CB is wanted to work full range.

4.2.1 Fuel Request and Engine Speed

Depending on the engine point the CB control strategy may be enabled or not, in Figure 4.2.1 it is represented a schematic that gives an overview on the logic with which the CB is enabled or not depending on the engine point. If the engine operating point is found to be inside the indicated boundaries the control strategy is enabled and the control delivers corrections.

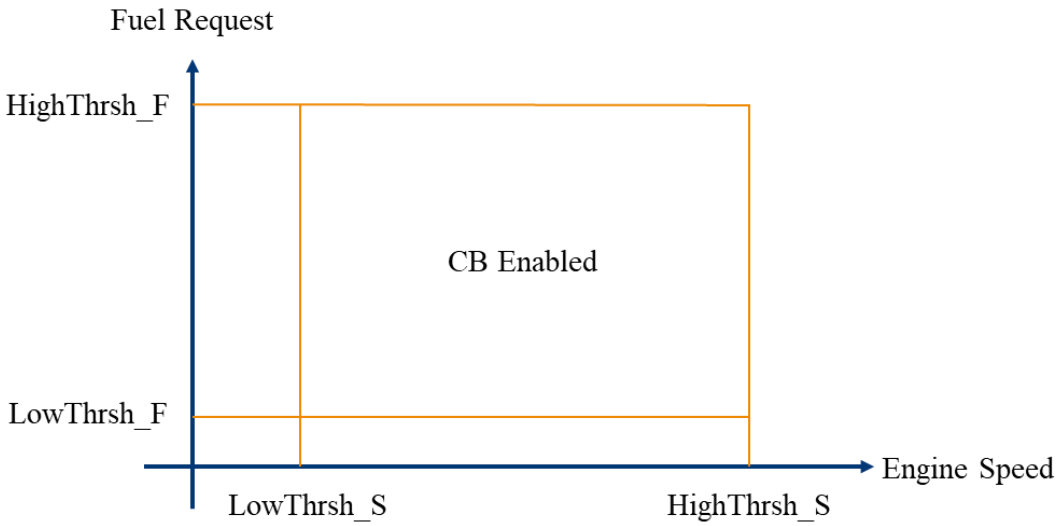


Figure 4.2.1: Fuel Request and Speed Enabling Conditions

The enabling boundaries, in terms of speed and fuel request must be calibrated on the specific application. The degrees of freedom are represented by the four calibration labels listed below:

- **LowThrsh_S**: Lower speed threshold
- **HighThrsh_S**: Higher speed threshold
- **LowThrsh_F**: Lower fuel request threshold
- **HighThrsh_F**: Higher fuel request threshold

The lower thresholds of fuel and speed are calibrated starting from the acquisitions on the engine running at idling. Once the speed at idling is acquired the lower threshold for the engine speed is set as the acquired one minus $50\,rpm$, while, the fuel request lower threshold is calibrated as the fuel request at idling subtracted by $1\,mm^3$ in order to be sure to have the CB enabled in all situations. The higher thresholds for speed and fuel request are calibrated in different ways, the first one is calibrated by taking into account the ECU Overrun Speed.

The Overrun Speed is the engine speed at which the ECU computational capability cannot catch up with the computational speed required. This means that the ECU above a certain speed is not able to deliver results or the results delivered may not be correct. The upper threshold for speed is then set as the value of the overrun speed subtracted by $50\,rpm$, this provides a safety margin even by enabling the CB in a wide speed range, almost full range. In Figure 4.2.2 is shown how the CBE reacts when the overrun speed is reached,

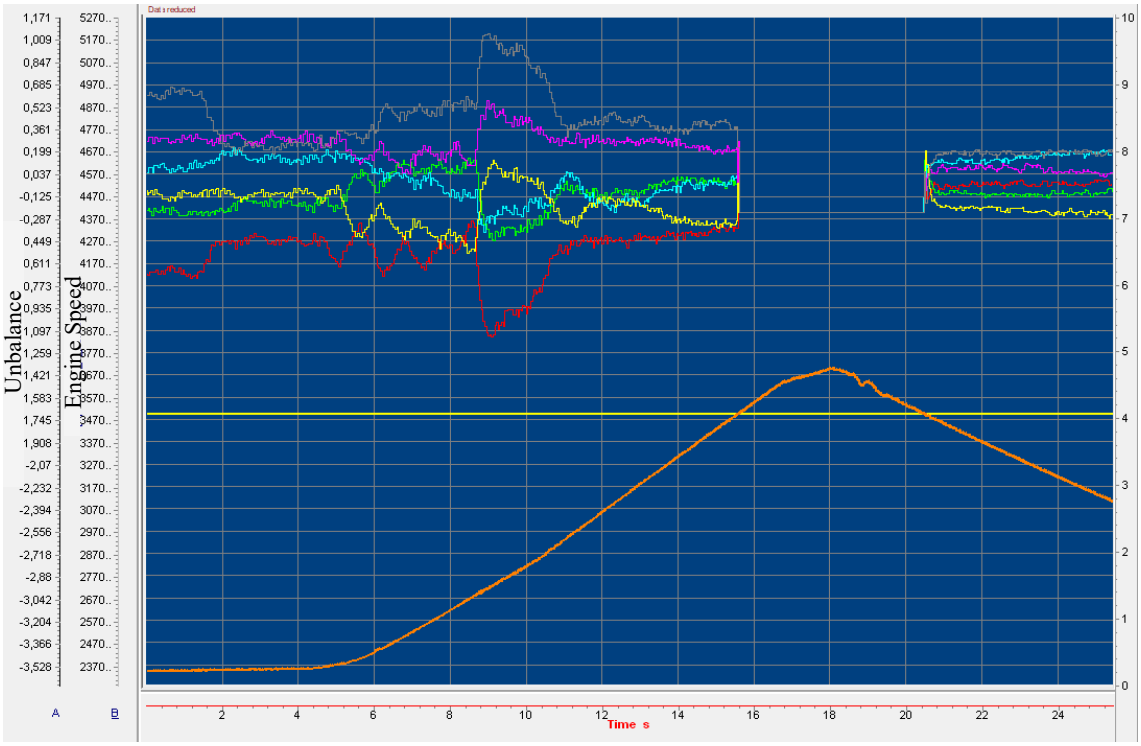


Figure 4.2.2: Overrun speed CBE shut off. Engine speed (Red Line) Overrun speed (Yellow Line).

indeed, when it happens the unbalance signals suddenly drop to zero since the CBE is turned off.

At last the Higher threshold for fuel request is not a single value but it depends on the engine speed, meaning it is an array of thresholds in which each value in the array is dependent by the speed.

In the original calibration procedure the engine was run at all engine speeds and maximum load in order to fill the array with the values of fuel requests at those engine points, since the fuel request at maximum achievable load must be the maximum achievable by the engine. During the review it was tried to think to an alternative way of calibrating the map without the need of new tests. Some Tests on torque delivery were already available from other teams which performed similar ones for other purposes. From these tests it is made available a map containing the fuel request at different engine speeds and torque, expressed in mg: see Table 3.

Table 3: Torque to Fuel map (Partial).

rpm/Nm	**0**	**25**	**50**	**75**	**100**	**125**	**150**
800	0	2	4	11	25	34	42
1000	0	2	4	11	24	31	39
1250	0	2	5	11	22	30	39
1500	0	3	5	11	22	30	38
1750	0	3	5	11	21	29	36
2000	0	3	5	11	21	29	35
2250	0	4	6	11	22	29	35
2500	0	4	6	11	23	30	38
2750	0	4	6	12	23	30	35

Once the previous map is available, to get the higher fuel request threshold one has to get the column of maximum torque in Table 3 and the values in that specific column correspond to the maximum fuel request thresholds. Notice that these values of fuel request are expressed in mg, they have to be divided by the fuel density in order to obtain the thresholds in mm^3.

4.2.2 Driveline Groups

Depending on the gear selected, the state of the torque converter and the state of the clutch, the software allows to group the states resulting from the combinations of these three factors into the so called Driveline Groups following the logic in Figure 4.2.3.

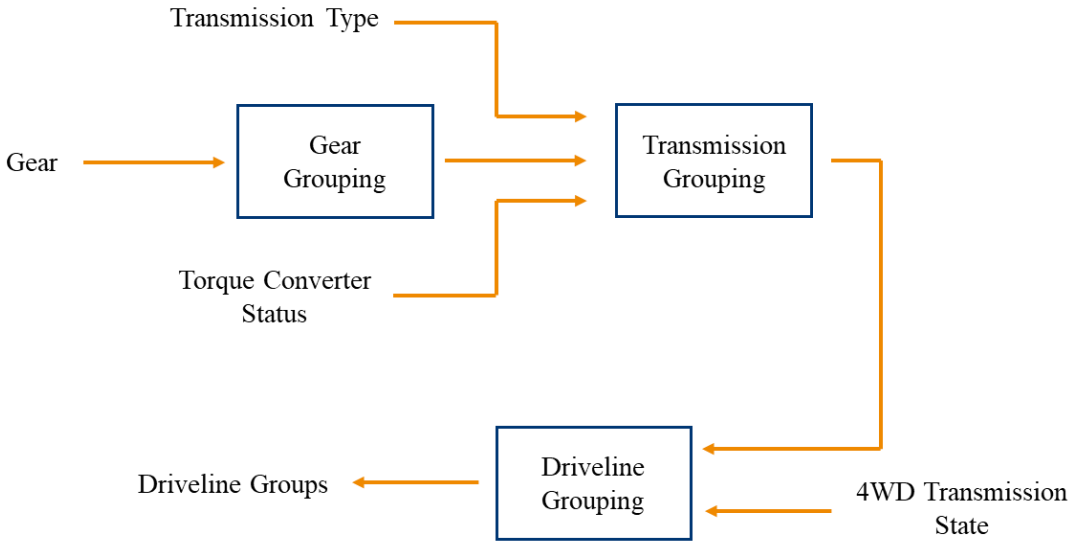


Figure 4.2.3: Driveline Grouping.

This feature allows to have more flexibility on the CB enabling, for example for some combinations of gears, traction converter status and clutch status the CB could be wanted to be off, but the CB is now wanted to be active in all situations. All Gear Groups are then calibrated in order to enable the CB at any combination of the three states.

4.2.3 Combustion mode

The combustion modes allow to group different injection patterns and decide whether enabling the CB control on a specific pattern or not.

Same as what happens for the Gear Groups, also the combustion modes are calibrated in order to have the CB enabled in all situations.

4.3 Correction conversion

In Chapter 3.3.5 it was highlighted how some kind of injectors have the need to be fed with corrections expressed in terms of energizing times ET, it was also explained how to get to the table that performs the conversion of the corrections.

The steps adopted in the previous calibration procedures provided to obtain the conversion table through calculations performed on excel, in the reviewed procedure, a Python tool was built in order to automate this process and eliminate the Human factor from these steps.

The conversion gains table is obtained from the Injector Map by performing the calculation specified in the equation below at each breakpoint of the Injector Map Table 1. The results of this operation are graphed in Figure 4.3.1, the plot shows how the $K_{Conversion}$ behaves at different fuel requests and constant pressure (each line correspond to a specific pressure value i.e. one column in Table 2).

$$K_{conversion}(1mm^3@20Mpa) = \frac{ET(1.5mm^3@20MPa) - ET(1mm^3@20MPa)}{1.5mm^3 - 1mm^3}$$

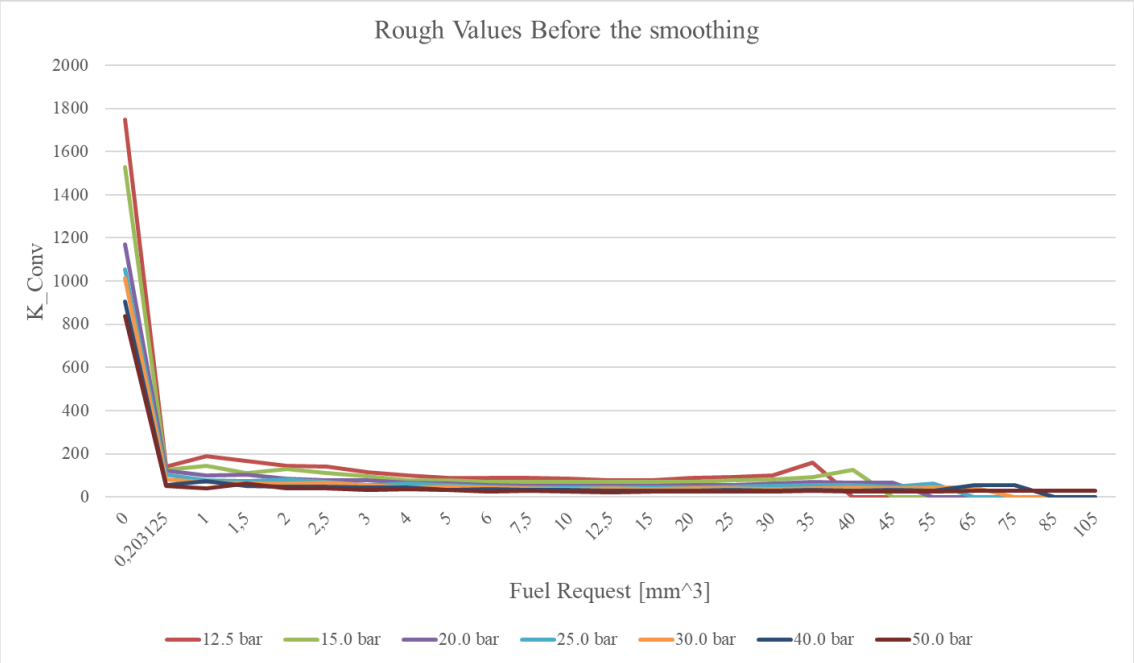


Figure 4.3.1: Results of the calculation through Python before the smoothing.

The data obtained from the previously described calculations need to be smoothed, local maxima need to be eliminated since they would lead in having a misleading conversion of the corrections, this aspect will be clarified later.

The smoothing algorithm (full code in chapter 6) that was decided to be adopted can be resumed as follows:

1) Select a window of points among the points at constant pressure
2) Calculate the line that best fits the points in the window
3) Check where the points are with respect to the fitted line, if they are lower than the line set the value of the points equal to that of the line, if they are higher than the line leave them as they are except for the case in which $Point\_value > Threshold * Line\_value$, in this case the value of the point is set equal to the line (i.e. the point is way higher than the line).

This procedure is applied in order to smooth the behavior of the curve and giving a decreasing trend of the conversion gains at constant pressure, over increasing fuel request. To be sure to have a decreasing trend over the whole curve a further step was performed. At each pressure the conversion gains vector is analyzed as follows. Starting from the end of the vector (highest fuel request) the value of each Gain is checked with respect to the previous and if it appears to be lower than that, it is set equal to the previous (full code in chapter 6). This step is needed to avoid having local minima in the curve, since it may lead in having a lower energizing time at higher fuel request at same pressure. For consistency the same check was applied by moving on constant fuel request row in order to avoid intersections between lines at constant pressure. These last steps are clarified at the end of the chapter.

In Figure 4.3.2 it is shown the initialization page that a user will make use of, from there the window of points can be selected, together with the threshold and the output name of the map. When it is run a preview plot is made and the smoothing can be evaluated, after that when the plot is closed the excel file is created.

At this point it can be highlighted why it is important to have a decreasing trend of the conversion gains at increasing fuel request. By taking a look at Figure 4.3.3, where the manual smoothing results are presented, it is clear that there are some inconsistencies. For example by analyzing the point around Fuel Request = 3 mm^3 one can notice that the energizing time decreases for some increasing fuel requests, this is not coherent with the theory and hence is not acceptable. Moreover, if the point around Fuel Request = 20 mm^3 is analyzed it can be noticed how for example at some point the energizing times on the 20 bar pressure line lie below the energizing times of the 25 bar pressure line. This is not acceptable since at increasing rail pressure at same fuel request the energizing time is expected to be lower.

```
WINDOW_SIZE = 5
THRESHOLD = 1.5
FILE_NAME_INPUT = "Excel_InjTbl/Injector_Map.xlsx"
FILE_NAME_OUTPUT = 'Excel_InjTbl/KtFULC_K_ET_CB_InjTblSlope_w5.xlsx'


def main():

    data_smoother = smoother(
        FILE_NAME_INPUT, FILE_NAME_OUTPUT, WINDOW_SIZE, THRESHOLD)
    # This object gives me back the excel file with the gains calculated
    data_smoother.smooth()
    data_smoother.grapher()

    ######### Here it is plotted the trend of the energizing times ET ############

    Smoothed_InjTblM = np.array(pd.read_excel(FILE_NAME_OUTPUT))

    Resultant_ET = Smoothed_InjTblM

    for j in range(1, Smoothed_InjTblM.shape[1]):

        for i in range(0, Smoothed_InjTblM.shape[0]):

            Resultant_ET[i, j] = Smoothed_InjTblM[i, 0]*Smoothed_InjTblM[i, j]

    plt.figure()

    for j in range(1, Resultant_ET.shape[1]):
        plt.plot(Resultant_ET[:, 0], Resultant_ET[:, j])

    plt.title('Fuel_Request*K_Conv @ Constant P --> ET')
    plt.grid("minor")
    plt.xlim([0, max(Resultant_ET[:, 0])-10])
    plt.xlabel('Fuel Request [mm^3]')
    plt.ylabel('K_Conv')
    plt.show()


if __name__ == "__main__":
    main()
```

Figure 4.3.2: Tool Interface

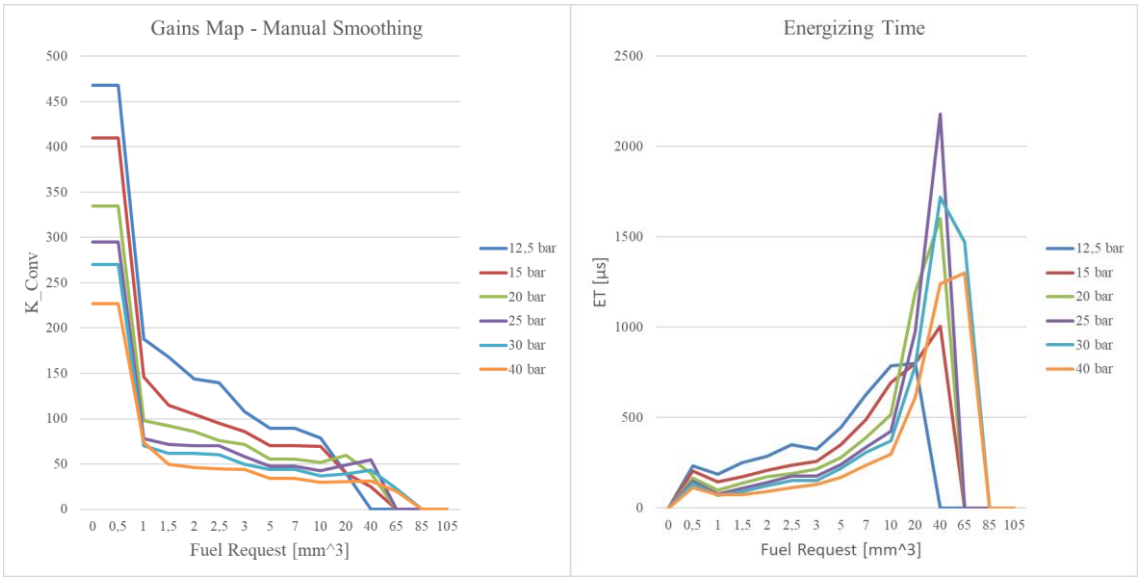


Figure 4.3.3: Manual Smoothing results

These are the reason why the smoothing algorithm has been designed as described, the results that are get through the python tool are shown in Figure 4.3.4. The trends of the curves are now coherent with the theory and can be available in a click time. Differences between the manual smoothing and the python smoothing are due to the human factor that is present in the manual smoothing. Indeed, to manually calibrate the map what was done was changing the values of the map in order to get a decreasing trend of the constant pressure lines by evaluating the trend visually, instead, with the python tool, mathematical theory is adopted making the process reproducible and consistent.

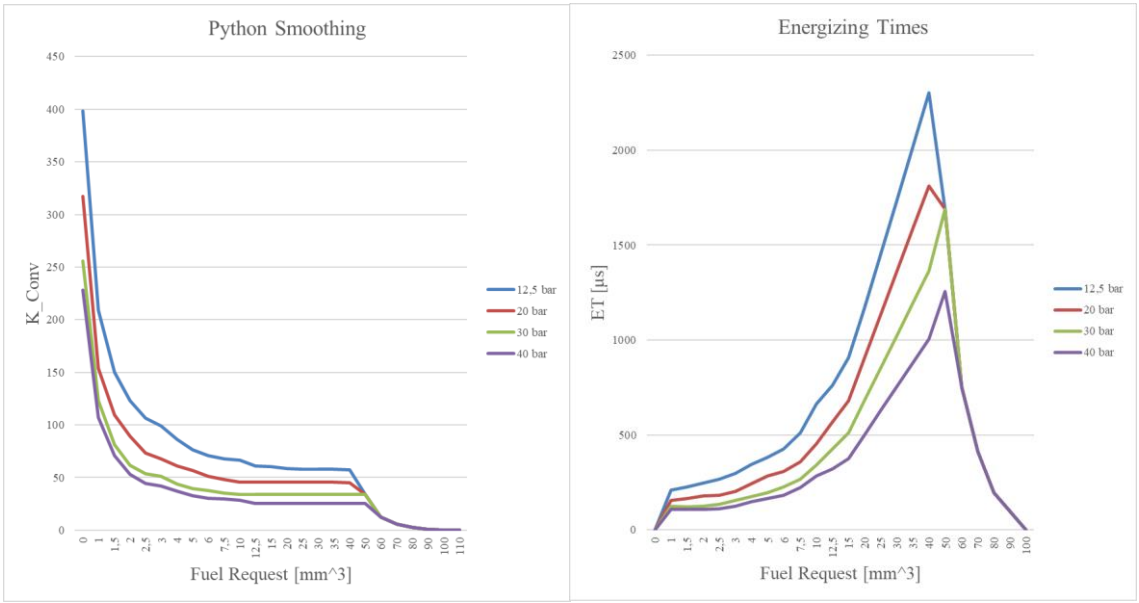


Figure 4.3.4: Python Smoothing Results

In Figure 4.3.5 is highlighted a comparison between: the original gains, manually smoothed gains and python smoothed gains lines at the same pressure (20 bar). As it can

be seen the python tool not only provides a line that is perfectly comparable to the manually smoothed but it also fits the original line in a better way, moreover, this result is obtained in a click time.

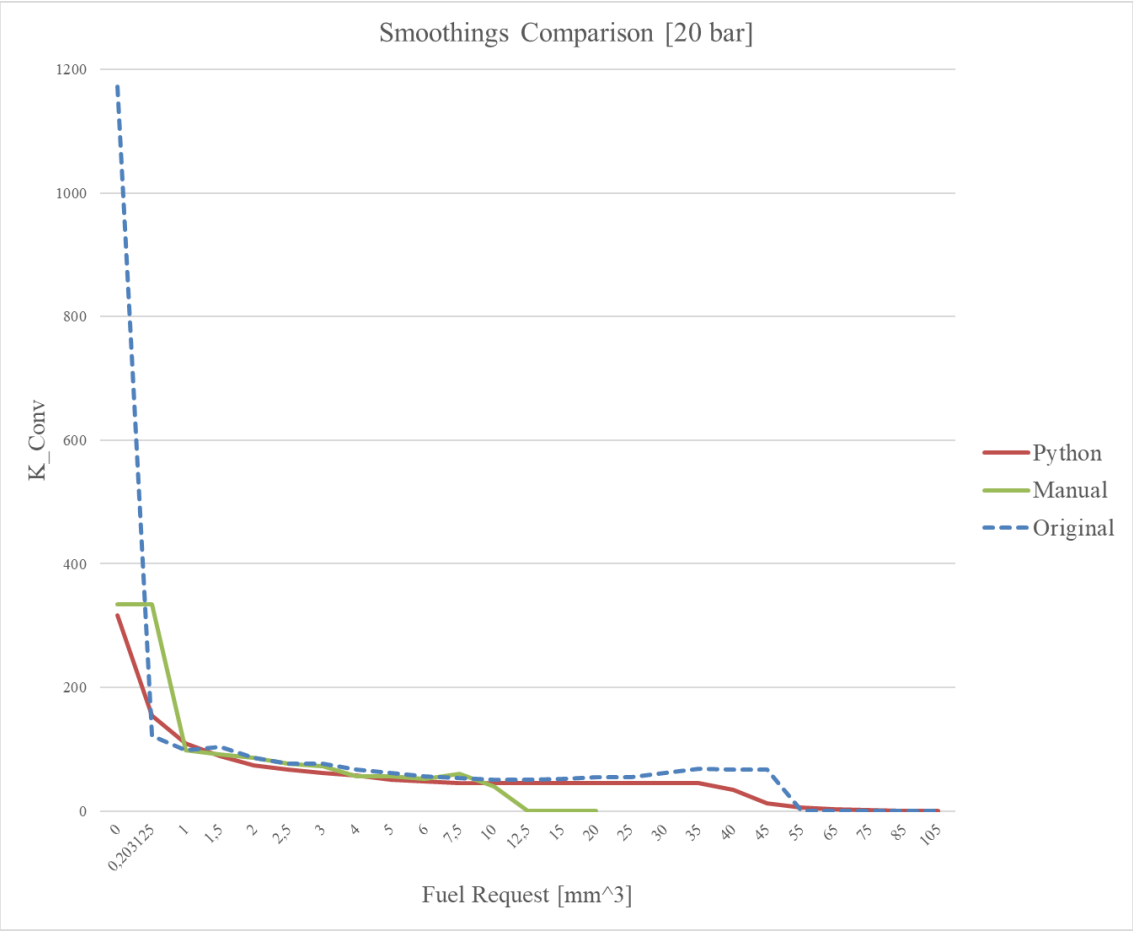


Figure 4.3.5: Comparison between the manual smoothing and the python smoothing.

4.4 Crank Angular Window Assignment

It was highlighted in Chapter 3.3.1 how the unbalance phenomena is referred to the cylinder responsible for its production. It was also specified how at different engine operating points the way with which the unbalance is referred to a specific cylinder should be adapted. Now it is quickly reminded which are the labels that need to be calibrated during this sub procedure:

- Map that specifies the assignment type, each row represents a different way of assigning the SubLores to the respective cylinder, Figure 3.3.3.
- Breakpoints of the map that specifies what assignment type to use at a certain engine point (i.e. which row to use in the previous calibrated map).
- Map that specifies what assignment type to use in between the previously determined breakpoints, Figure 3.3.4.

The original calibration procedure provided to test, at **steady state**, different engine points and from the observation of the unbalance signals, on INCA, determining whether a certain type of assignment works in a specific engine point or not.

An example of test that was performed consisted in running the engine at 1500 rpm, imposing a fuel request of 15 mm^3, calibrate the assignment type in one way (i.e. AABBCCDD…), drifting one cylinder and by looking at the unbalance signals determine whether the assignment is correct or not. One of the drawbacks of such a procedure is that the engine points are tested at steady state while the engine operates most of the times in **dynamic**. Moreover, on the available roller test bench, the braking torque capability of the roller bench, allows to reach, at steady state, fuel request of about 40 mm^3, while the maximum reachable by the engine under considerations is about 100 mm^3, meaning that a wide fuel to speed area cannot be tested with the former procedure.

It was decided to try a different approach in order to solve the previously presented issues and make faster the calibration procedure. During testing it came out that the roller test bench can withstand the torque produced by the engine with a fuel request of about 90/100 mm^3 in dynamic conditions, meaning that if those fuel requests are met for a few instants the roller test bench does not incur in roller slip as it happens at steady state operation at the same fuel request. These observations led to develop a procedure that provides to perform many tests in transient condition and feed the results to a python tool that helps in managing the results to get to a final calibration in an easier and more efficient way.

4.4.1 Test Matrix

The tests to be performed, according to the new procedure, are resumed in Table 4. These have to be performed with the same assignment type in the whole working area and the unbalance signals have to be acquired, together with engine speed and fuel request.

Table 4: Test matrix for cylinder assignment

Test	Starting Conditions	Stop Conditions	CB	Drift	Gear	Dyno Slope	Pedal Position
1	FTP72 – 6 min		OFF	$+3\sigma A$	NA	0%	NA
2	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	5%	60%
3	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	10%	60%
4	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	15%	60%
5	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	5%	100%
6	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	10%	100%
7	Idle Brake On	CBE Overrun	OFF	$+3\sigma A$	1st to allowed	15%	100%

At this point it is important to explain some terminology adopted in the test matrix in Table 4:

- **Starting Conditions**: is the condition from which the acquisition is started. The state "Idle Brake On" represents a situation in which the car is still, the first gear is engaged and the brake pedal is pushed.
- **Stop Conditions**: is the condition at which the acquisition must be stopped. The condition "CBE Overrun" means that the engine speed at which the ECU overrun occurs, has been reached.
  It can be noticed that, for test 1, it has been indicated both for start and stop condition "FTP 72", this is because the FTP 72 is a kind of cycle that is performed to evaluate emissions but it is suitable also for the current scope and does not really have a marked start and stop condition, it just have to be performed for 6 minutes.

- **CB**: it indicates the CB status, for the current purpose the CB must be off in all tests since what is tried to be evaluated is the assignment and the full CB control system has not been calibrated yet.
- **Drift**: it indicates what is the magnitude of the unbalance that has been applied to one cylinder by drifting its respective injector. The drift is expressed in sigma σ which is the amount of deviation from the nominal condition, it is applied to the injector through a strategy named IFI.
  The IFI allows to deceive the injector map simulating that the injector is aged. A drift of $+3\sigma$ is the worst case acceptable for an injector, the IFI map that determines this ageing is retrieved after the so called General Engine Durability test (GED), in which an engine is brought near its end of life and the injector performances are evaluated. For all tests it has been applied a positive drift on injector A.
- **Gear**: it specifies in which gear the test has to be performed, for the tests specified the gears are not decided by the driver but by the ECU depending on the operating point. The operator should just press the pedal and hold it at the specified position.
- **Dyno Slope**: the roller test bench can simulate road inclination, this is expressed in percentage. It was varied during the tests in order to make the fuel request vary and so being able to cover more engine operating points.
- **Pedal Position**: it indicates how much the accelerator pedal has been pushed, as for the dyno slope, this is adopted in order to vary the fuel request.

Are shown in Figure 4.4.1, Figure 4.4.2 and Figure 4.4.3 the acquisitions of some of the tests illustrated in Table 4. Figure 4.4.1 shows test number 3, where the dyno inclination is 10 % and the pedal position is 60 %, the maximum fuel request reached is about 55 mm^3. Figure 4.4.2 shows test number 6 in which the dyno inclination is 10 % and the pedal position is 100 %, indeed, in this test the maximum fuel request reached is about 95 mm^3. In the end Figure 4.4.3 shows the first 6 minutes of a FTP 72 test, during this kind of test the engine points that are more likely to be visited during a normal drive are met (i.e. low fuel request). Indeed, this kind of test is performed in order to get more densely distributed set of points in the working area that is covered most of the times.

When all the tests listed in Table 4 are performed a wide part of the working area of the engine will be covered, in Figure 4.4.4 all the engine points that have been visited during the tests are shown in blue, from the figure it is clear how a wide area of the whole working area can be now evaluated.

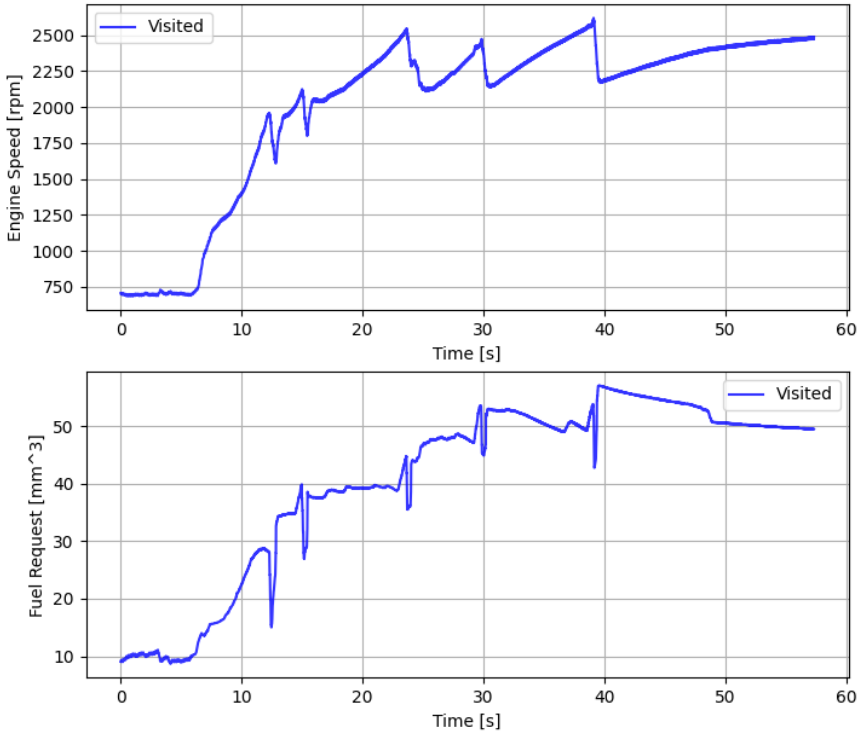


Figure 4.4.1: Test 3, Pedal Position 60% and Dyno Slope 10%

Since the maps that have to be calibrated depend on the engine point (fuel request and engine speed), if it is highlighted, in Figure 4.4.4, in which points the assignment is not correct (these points will be referred to as Wrongly Assigned Points WAP) this would make the calibration process instantaneous just by looking at Figure 4.4.4 with the addition of WAP.

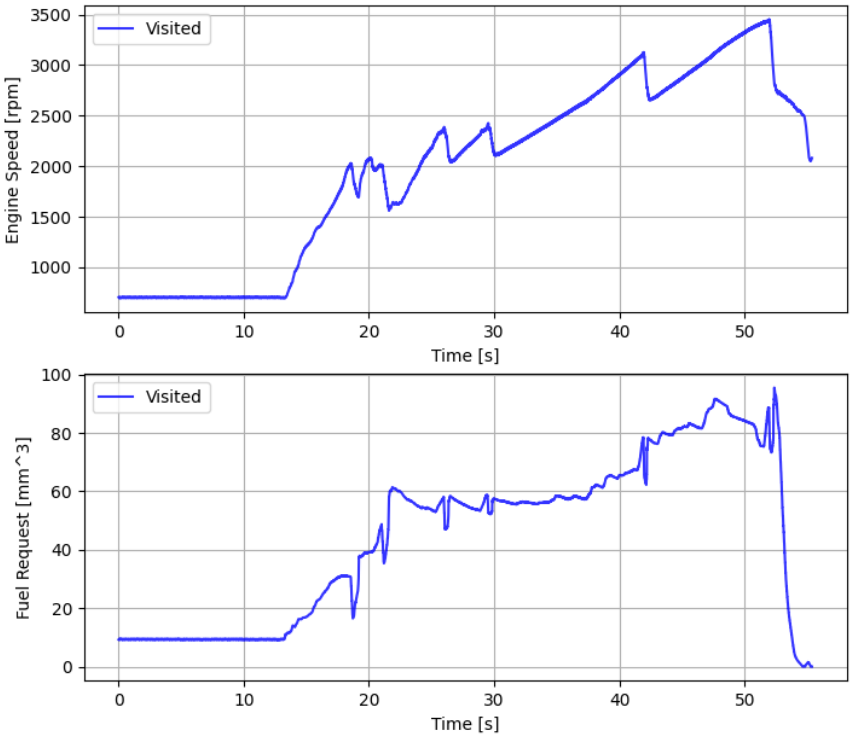


Figure 4.4.2: Test 6, Pedal Position 100% and Dyno Slope 10%

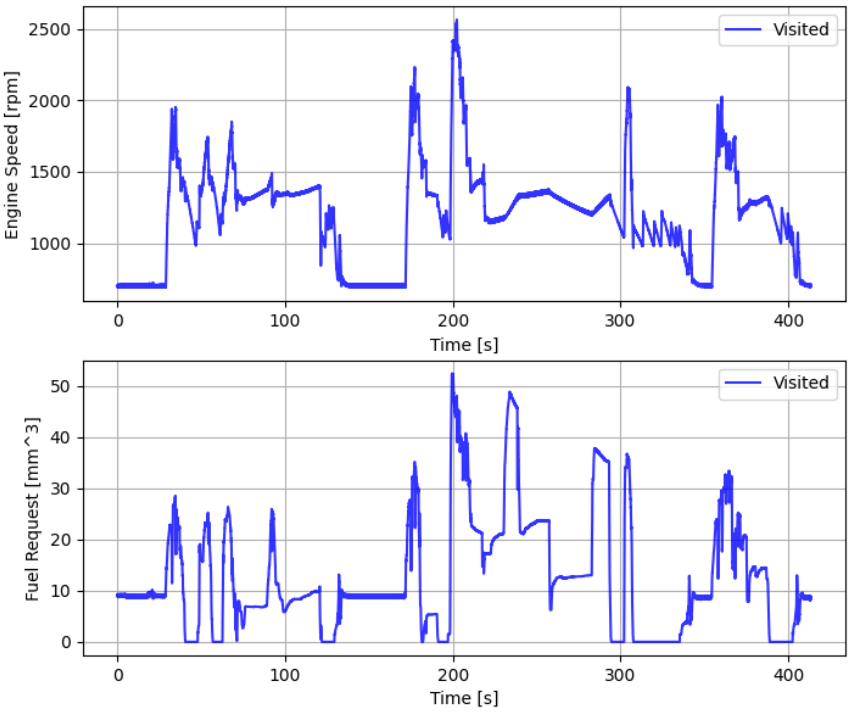


Figure 4.4.3: FTP 72 lasting 6 minutes

What the tool does is indeed getting the unbalance signals of all tests as input and checking whether the unbalance signal of the drifted cylinder is coherent with the drift

applied. If the unbalance signal is not coherent with the cylinder that was drifted, in a certain engine point, that engine point is addressed as WAP and stored.

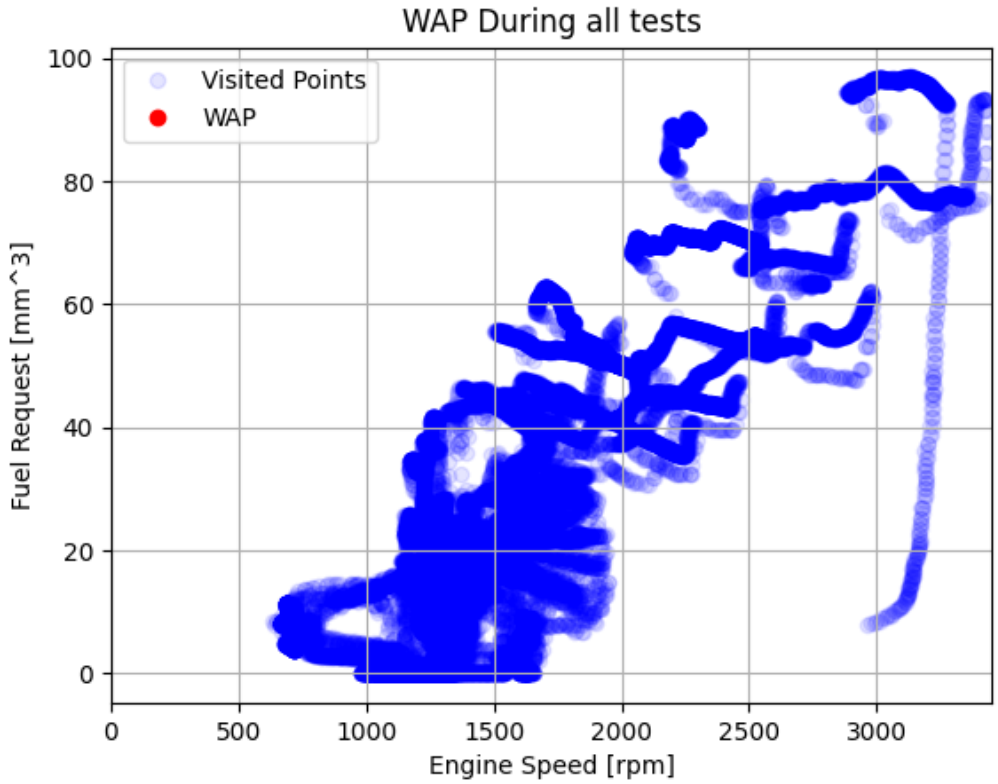


Figure 4.4.4: Engine points visited during the tests.

4.4.2 CAWA Python Tool

In chapter 3.3.2 it is highlighted how, in a balanced engine, when a **positive** drift is applied to a single cylinder, that cylinder should produce the lowest unbalance signal which would translate into a **negative** correction. If the drift applied is negative, the opposite situation would occur.

What the tool does in order to determine whether the cylinder assignment is correct or not, is checking, sample by sample, if the unbalance signal relative to the drifted cylinder is the lowest/highest (depending on the type of drift +/-) than all the other unbalance signals. If this condition is not met the tool saves at which engine point this occurs and it is addressed to as a WAP.

In Figure 4.4.5 it is shown the tool initialization section, from there it is possible to set the parameters, depending on the application, the full code can be found in Chapter 6. The parameters are:

- **UNBAL_CYL**. Cylinder to which the drift was applied during testing.
- **TYPE_OF_DRIFT**. Can be either Positive or Negative.
- **NUMBER_OF_CYLINDERS**. The tool was designed to work with up to 8 cylinders engines.
- **MAX_SPEED_OVERRUN**. Overrun speed of the engine.
- **LOWER_FUEL_THRESHOLD**. Threshold below which it is not wanted to perform the check on the unbalances.
- **TOLERANCE**. Imagine having a positively drifted cylinder i.e. Cyl_A that produces a negative unbalance signal with magnitude 3. If the lowest negative unbalance signal is not the one referred to the drifted cylinder but for example is referred to Cyl_B, the engine points is assigned as WAP only if $Unbal_{cylA} > (Unbal_{cylB} + TOLERANCE)$. This was added to remove noises in the signal.
- **DEAD_BAND**. Is the value of the dead band under which the corrections no more change and so the unbalance signal is not influent.
- **GEAR_SHIFT_NEGLECTION**. It is possible to decide if the window in which a gearshift occurs should be neglected or not by setting this variable to True, otherwise to False, remember that the variable relative to the selected gear should be fed to the tool. It can be useful to observe just the situations in which there is no variability due to a gearshift which induces a strong transient.

```
##############################----------------INITIALIZATION SECTION---------------##################################

# This is the cylinder i have drifted range from A to H (8 Cylinders maximum)
UNBAL_CYL = "A"
# TYPE_OF_DRIFT --- May be either "Positive" or "Negative"
TYPE_OF_DRIFT = "Positive"
# NUMBER_OF_CYLINDERS --- Number of cylinder in the engine
NUMBER_OF_CYLINDERS = 6
# MAX_SPEED_OVERRUNN --- Maximum speed at which the CBE is active
MAX_SPEED_OVERRUNN = 3500
LOWER_FUEL_THRESHOLD = 2
# TOLERANCE --- If the drifted cylinder is not the highest/lowest point but still in this range from
# the highest/lowest point then its assignment is considered valid. This allows to clean the results
TOLERANCE = 0.1
DEAD_BAND = 0.5
# GR_SHIFT_NEGLECTION --- This is a boolean and is set either True or Flase depending if it is wanted that a window of points
# is neglected during gearshifting, this allows to clean the results
GR_SHIFT_NEGLECTION = False
##################################----------------OUTPUT MANAGEMENT-----------------##################################
# The name following Acquisitions\Results\..... is the name of the excel file given in ouput containing a sheet for each test
FILENAME_OUTPUT = 'Acquisitions/Results/Res_neglected={}_GrShift_Tol_{}.xlsx'.format(
    GR_SHIFT_NEGLECTION, TOLERANCE)
```

Figure 4.4.5: CAWA Tool Interface

When the tool is fed with the tests specified in section 4.4.1 it returns the plot in Figure 4.4.6. The blue points represent all the engine points that were visited during all the tests, while the red points are the WAP. This plot is extremely helpful when having to determine which should be the break points of the map that decides the assignment type depending on the engine point, an example of thresholds was inserted in Figure 4.4.6 in red, in order to give an idea.

Remember that all the tests were performed with the same assignment type at any engine point, it means that in the blue area the specific assignment type is working correctly while in the red areas a different type of assignment should be applied. In the next chapter the step by step procedure to get to the calibration by means of the tool will be explained.

4.4.3 Calibration steps

As it was highlighted many times the tests for this procedure have to be performed with the same type of assignment in the whole working area. The type of assignment to adopt for these tests should not be random but determined before the tests in the following way.

With the engine running at idle, apply a positive drift to cylinder A and through INCA look whether the unbalance signal relative to cylinder A is the lowest. If it appears to be

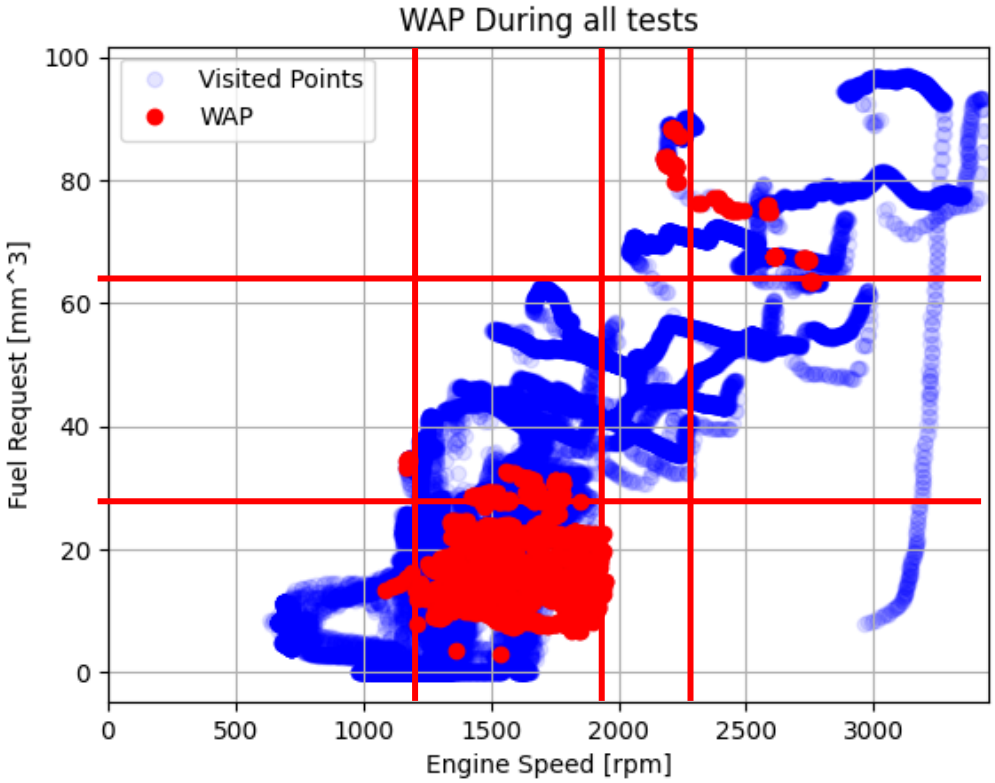


Figure 4.4.6: Visited Points together with Wrongly Assigned Points WAP.

true, then this type of assignment is the one to be adopted for the whole working area during the tests, since is the most likely to work in most of the areas. If in idle the assignment is not correct it should be shifted according to which cylinder is assigned

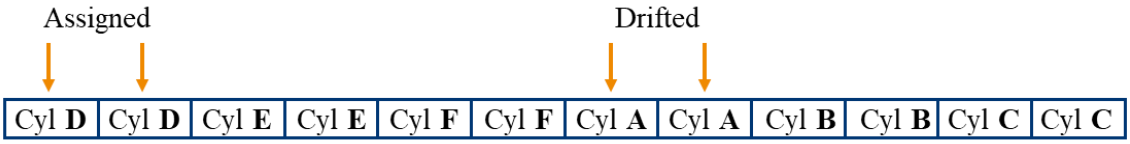


Figure 4.4.7: Wrong assignment at Idle.

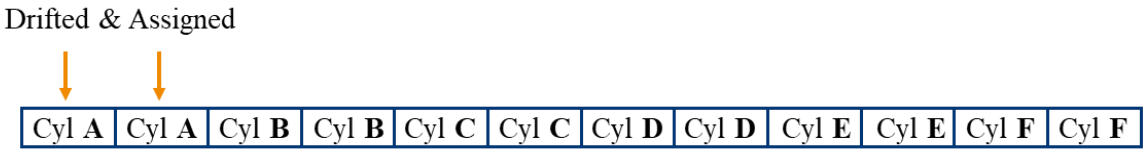


Figure 4.4.8: Correct assignment at Idle.

instead. For example, during the current calibration with the assignment type in Figure 4.4.7 while drifting cylinder A, Cylinder D was assigned instead, this means that the assignment should be corrected to get the one in Figure 4.4.8 which will be the one adopted for all the tests.

In order to have a clear idea on how the assignment should be calibrated during testing it is inserted a crop from INCA calibration manager, Figure 4.4.9. The upper window determines which row in the lower window is adopted in a specific working point. Since for testing the assignment is wanted the same in all points this is calibrated with all ones and the row one is calibrated with the procedure explained before.

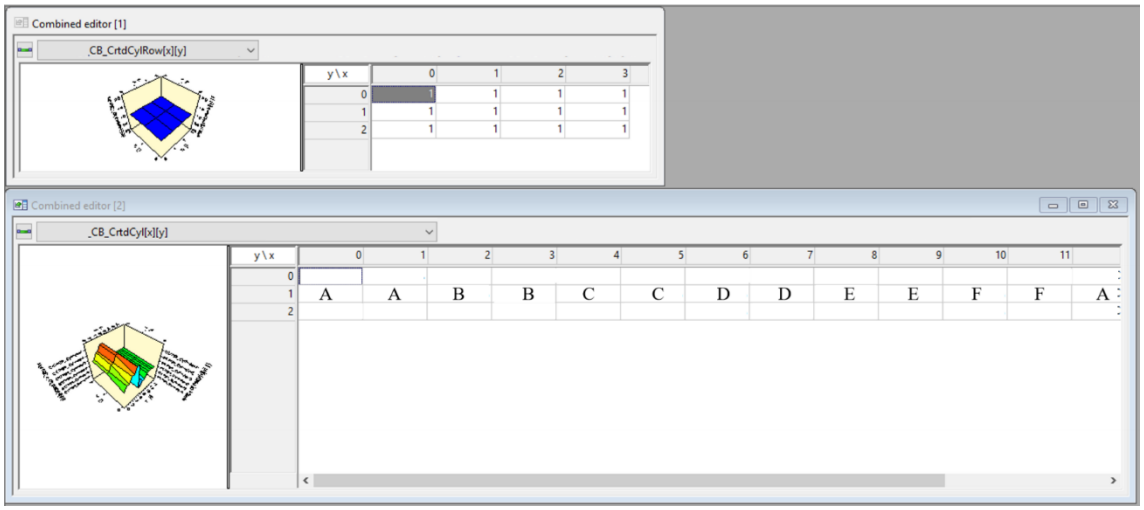


Figure 4.4.9: INCA Calibration Manager.

It is now possible to proceed with the specified test and acquire the unbalances, fuel request and speed that are later fed to the tool which returns the graph in Figure 4.4.6.

The tool identifies two areas in which the assignment seems to fail with the calibrations in Figure 4.4.9. One of the areas is the one highlighted in Figure 4.4.10, although in that area the assignment seems to fail in some points, from a deeper analysis available in the tool, it can be seen that only 101 points over 3859 were assigned wrongly, meaning that the current assignment is still the best performing.

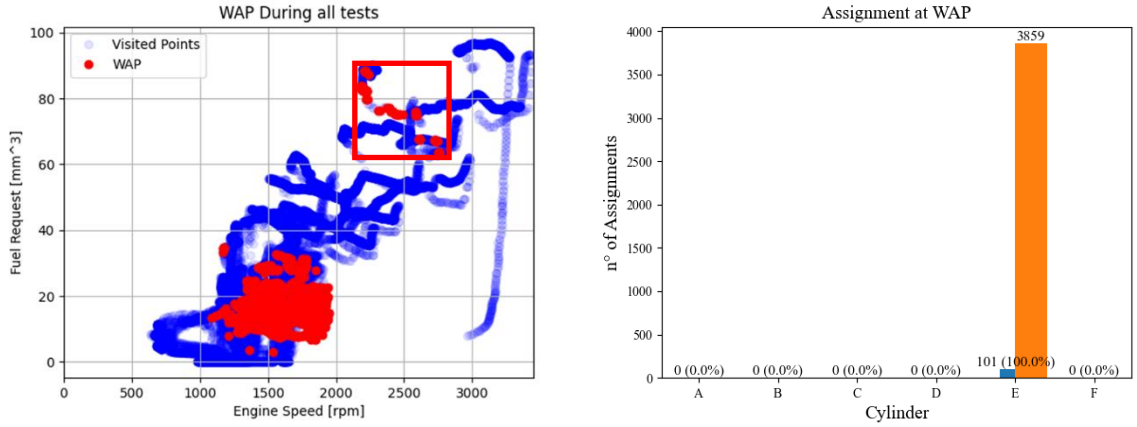


Figure 4.4.10: Upper WAP Area.

The second area in which the assignment is wrong is highlighted in Figure 4.4.11. Here the situation is a little different from the previous since by looking at the bar graph it is possible to see how on 10875 points visited, 1673 are wrongly assigned points, amounting at about the 15% of the total points.

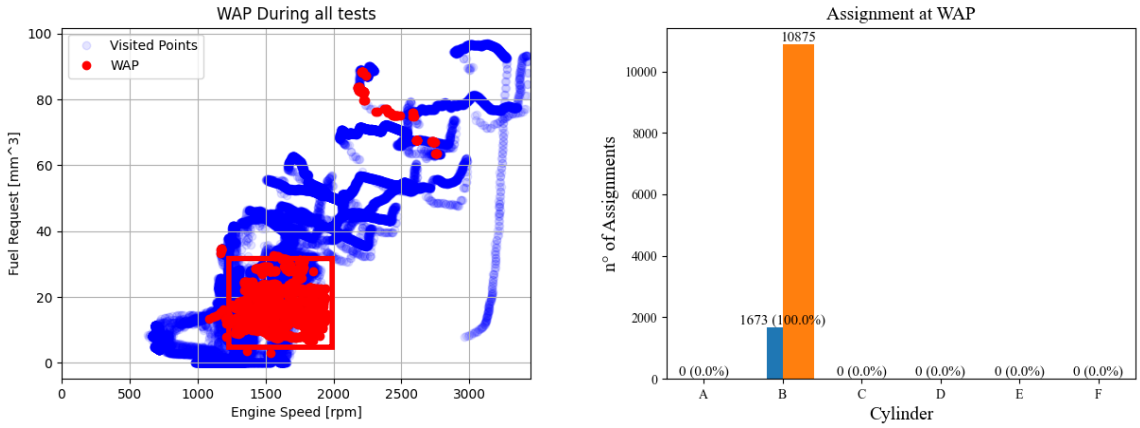


Figure 4.4.11: Lower WAP Area.

Another information that can be drawn from the bar graph analysis is that 100% of the times in which the assignment is wrong Cylinder B is assigned. This means that by forcing a different assignment in that area, through calibration, the assignment will be working correctly even in that area. Since the assignment type adopted during these tests is of the type in Figure 4.4.9 by shifting the assignment by one Sub-Lores toward Cylinder B, getting the assignment type in Figure 4.4.12, this problem is avoided.

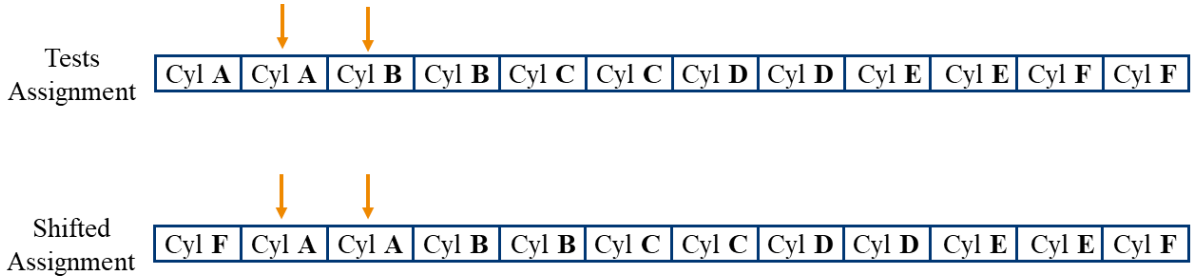
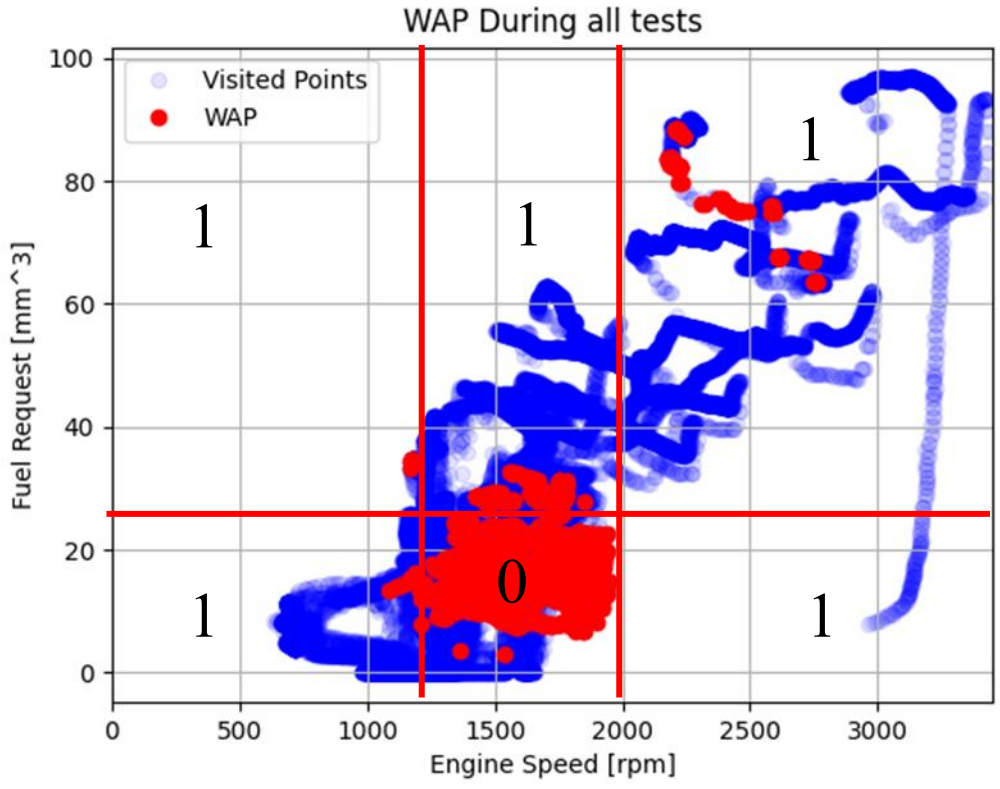


Figure 4.4.12: Shifted assignment for low Fuel & Speed Area.

The assignment is calibrated in such a way by setting the speed and fuel request thresholds and assignment types in order to isolate the red area as in Figure 4.4.13.



Ass. 0	Cyl **F**	Cyl **A**	Cyl **A**	Cyl **B**	Cyl **B**	Cyl **C**	Cyl **C**	Cyl **D**	Cyl **D**	Cyl **E**	Cyl **E**	Cyl **F**
Ass. 1	Cyl **A**	Cyl **A**	Cyl **B**	Cyl **B**	Cyl **C**	Cyl **C**	Cyl **D**	Cyl **D**	Cyl **E**	Cyl **E**	Cyl **F**	Cyl **F**

Figure 4.4.13: Full calibration of the assignment.

4.5 Integral Constants

In this Chapter it is described how to calibrate the map that contains the gains of the integral closed loop control of the CB.

The integral gains are selected depending on the fuel request and the engine speed, moreover, the integral gains adopted in steady state situations are different from the ones adopted in transients. Indeed, when a transient is detected, the integral gain becomes equal to $K_{trans} = K_{SteadyState} * K_{multiplicative}$. A multiplicative map must be calibrated ($K_{multiplicative}$) for each speed and fuel request breakpoint.

4.5.1 Steady state operation

In the original procedure the calibration of the integral gains is a very time and test demanding procedure. It was provided to calibrate the integral gains by running the engine at steady state at each engine point defined by the breakpoints of the Integral gains map. In each of the tested engine points it was needed to drift a cylinder and activate the CB in order to evaluate the convergence time, overshoots and oscillations of these signals.

The steps had to be iterated at changing gain until the unbalance signals reached a satisfying convergence to zero. Remember that all these tests were performed at the roller test bench and for each engine point at least 3 tests had to be performed, leading to a total amount of tests near 40/50 needed to calibrate this map.

Tests at the test bench are very cost demanding, since they require to have the whole vehicle available together with the resources to manage the facility. Reducing the number of tests to be performed at this facility is crucial, this is why it was thought of a different approach in order to calibrate the integral constants map with way less tests with respect to the ones required in the former procedure.

The new approach consists in estimating a transfer function that encloses what is the behavior of the unbalance signal when a drift is applied, at each engine point required from the gains map. Once the transfer function is available, by inserting it inside a simulated CB control system on Simulink, it is possible to simulate the behavior of the system at different integral gain values. In this way the convergence time, the overshoots and the oscillations of the unbalance signals can be evaluated at the desk by means of Simulink, avoiding many tests on the vehicle and making the procedure faster through automatization of the simulations.

In order to estimate what is the transfer function that encloses the behavior of the system when a drift is applied it is needed to perform a test in which the unbalance signal relative to the unbalanced cylinder is acquired. The test is started with the engine that is perfectly balanced, then, a drift is applied through the IFI and when the unbalance signal is steady the test can be stopped. An example of test is reported in Figure 4.5.1, where the engine is running at 1500 rpm and the fuel request is 10 mm^3. What is done is basically obtaining the step response of the system from which the TF will be retrieved.

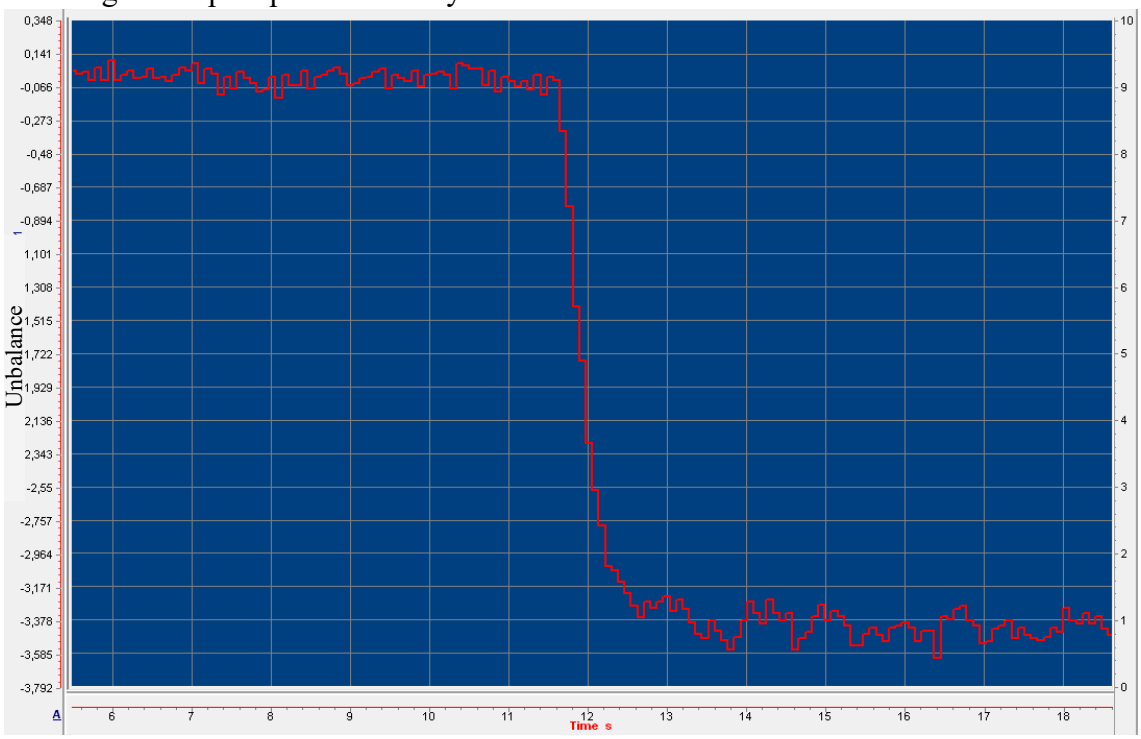


Figure 4.5.1: Cylinder A Unbalance signal when drifted +3sA, 1500 rpm 10 mm^3.

By using the System Identification toolbox in MATLAB it was decided the order of the transfer function to be estimated, Figure 4.5.2, and the parameters of the TF were determined through the *tfest()* command in MATLAB.

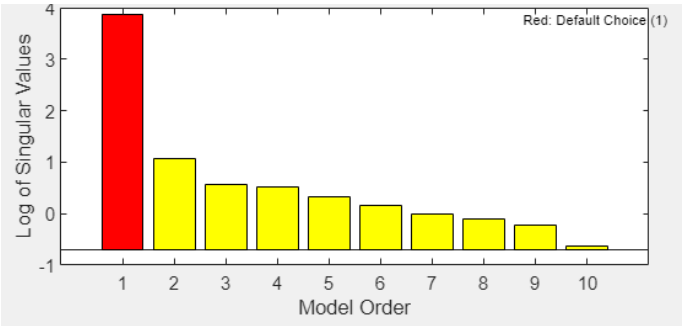


Figure 4.5.2: TF Order estimation through System Identification Toolbox.

Based on the results of the System Identification Toolbox it was decided to adopt a TF with two poles and one zero (second order TF) to simulate the engine response to the injector drift.

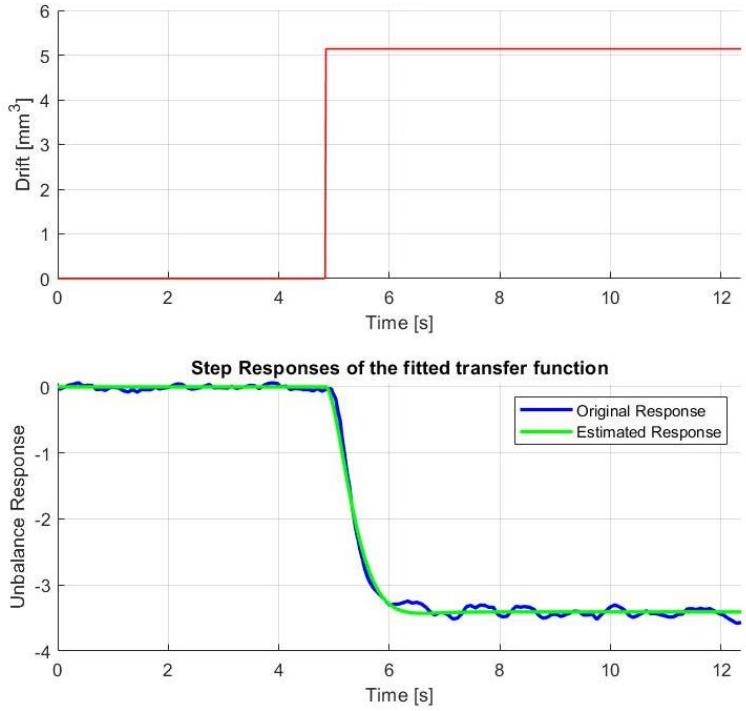


Figure 4.5.3: Step response of the system.

This kind of transfer function is adopted and fit to the tests, it was noticed by testing many engine points that the order of the transfer function that best fits the system is always 2. In Figure 4.5.3 it is shown how the estimated transfer function behaves compared to the original system when applying a drift. The fit to estimation data provided by MATLAB amounts at 96.58% with a Mean Squared Error of 0.003164. This suggests that the estimation of the TF is very good. Once the estimated TF is available it is inserted in a model that was developed to replicate the behavior of the real CB in order to be able to simulate what is the behavior of the whole system (i.e. TF + CB) at varying integral gains values. In Figure 4.5.4 it is shown the system in which the estimated transfer function is

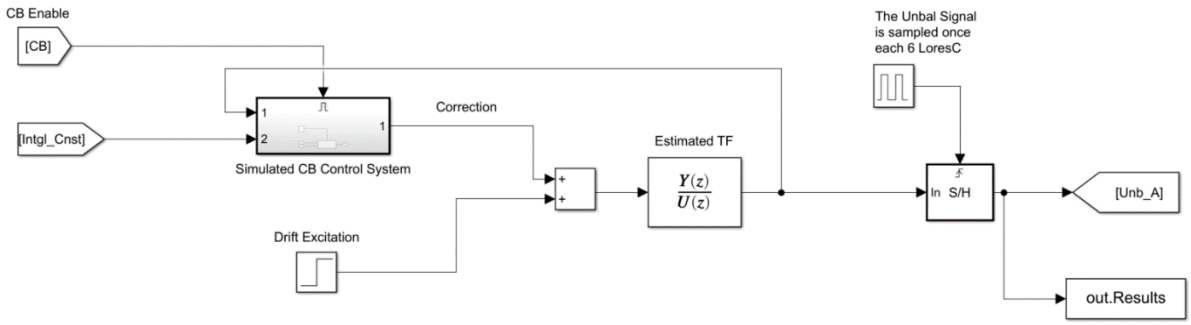


Figure 4.5.4: Simulated System.

inserted. The input to the TF is the drift excitation, summed with the output of the simulated CB Control System Figure 4.5.5, which is the correction at a given time instant, the output of the TF instead is the unbalance signal, which is to be evaluated at varying $Intgl\_Cnst$ values.

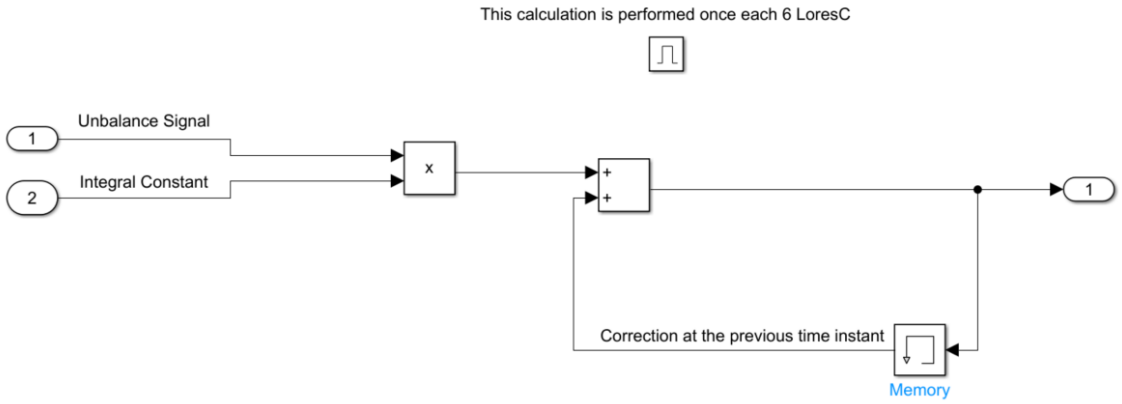


Figure 4.5.5: Simulated CB Control System.

Taking into account the presented test, at 1500 rpm and 10 mm^3 of fuel request and the integral gain equal to 0.1, the result of the simulation is presented in Figure 4.5.6. The green line represents when the drift is applied while the red line is the CB status, when it is equal to one the simulated CB is enabled. What has to be evaluated is the behavior of the unbalance signal (blue line) when the CB is enabled. This simulation was performed

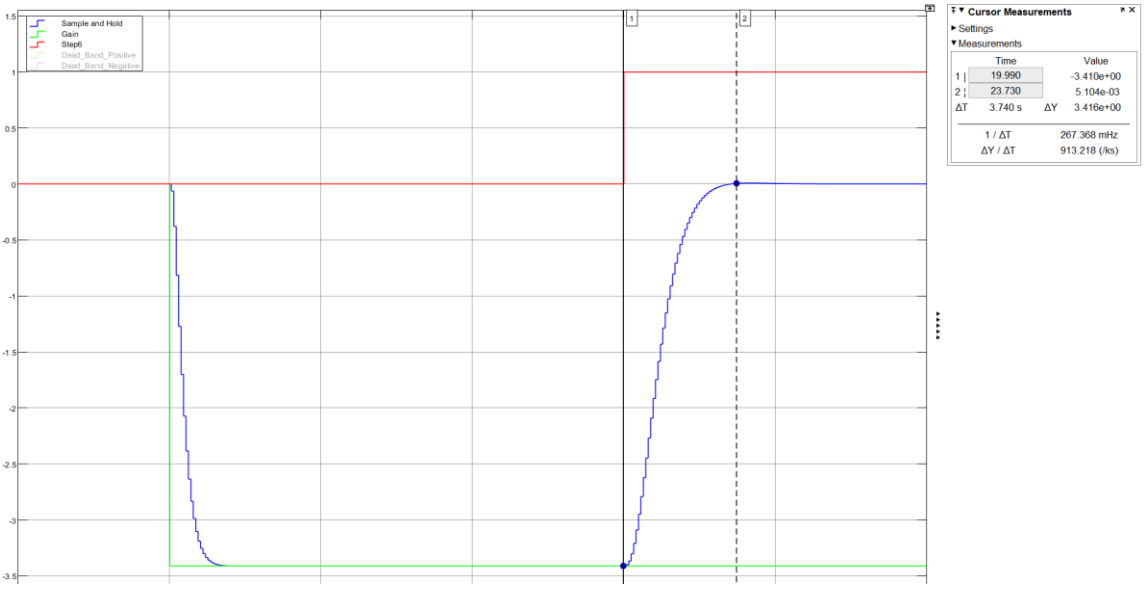


Figure 4.5.6: Simulation results for Intgl_Cnst = 0.1

with the same integral constant of the test in Figure 4.5.7 which was performed on the vehicle.

In terms of rise time and overshoot performances, the simulated unbalance is perfectly comparable to the real test performed at the dyno. The rise time in the simulation is around 3.74 seconds while in the real test is about 3.8 seconds (Figure 4.5.7), these results consolidate the theory ruling the tool. Another important aspect that needs to be considered to evaluate the goodness of the simulation is that, when the simulated CB is

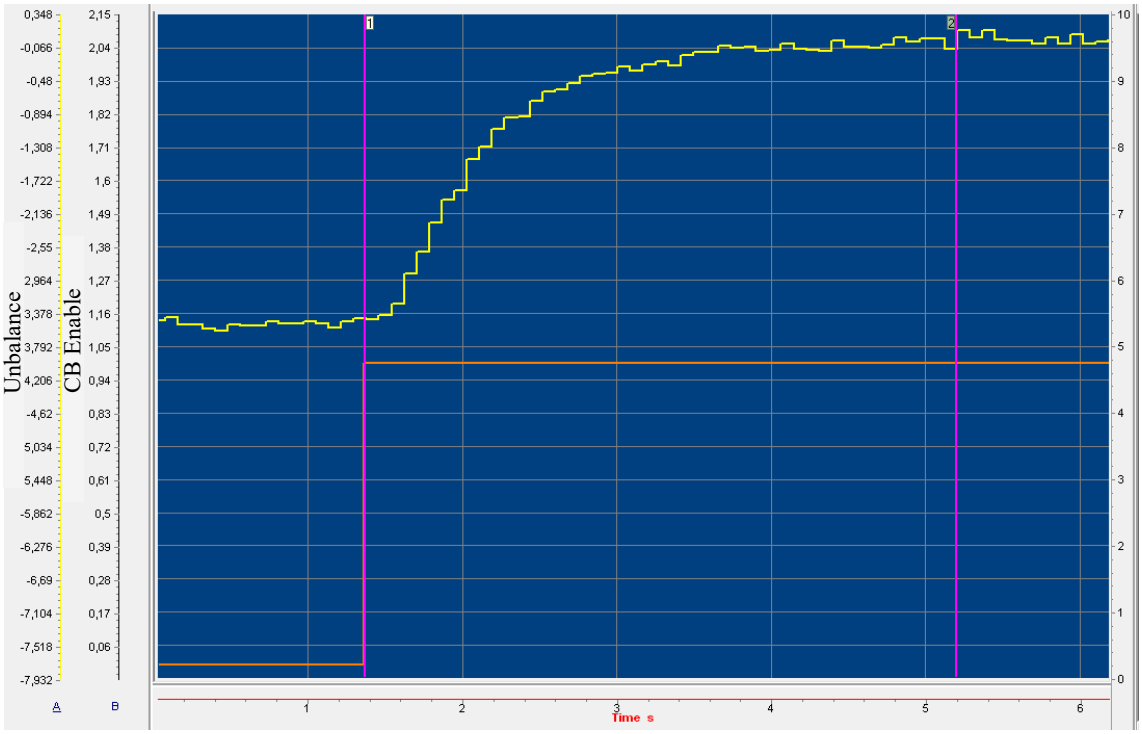


Figure 4.5.7: Test at the dyno, +3σA, Intgl_Cnst = 0.1.

not enabled one may say that it is obvious that the behavior of the simulation is comparable to the real test, since the TF was fit to it. However, when the simulated CB is enabled the TF is inserted inside a system that was modelled without fitting a transfer function to data but by modelling the real control system and the behavior of the simulation, when simulated CB is enabled, is perfectly comparable to the real system, meaning that the CB was modelled in a very accurate way.

The tool is then automatized in order to get to a result without having to cycle by hand through different integral gains. What the tool does is simulating different gains until the unbalance performances in terms of overshoot and rise time are met, Figure 4.5.8. In this way, with just one test for each engine point it is feasible to optimally calibrate the integral gains map for steady state. This procedure reduces the number of tests at least of 1/3 with respect to the previous procedure.

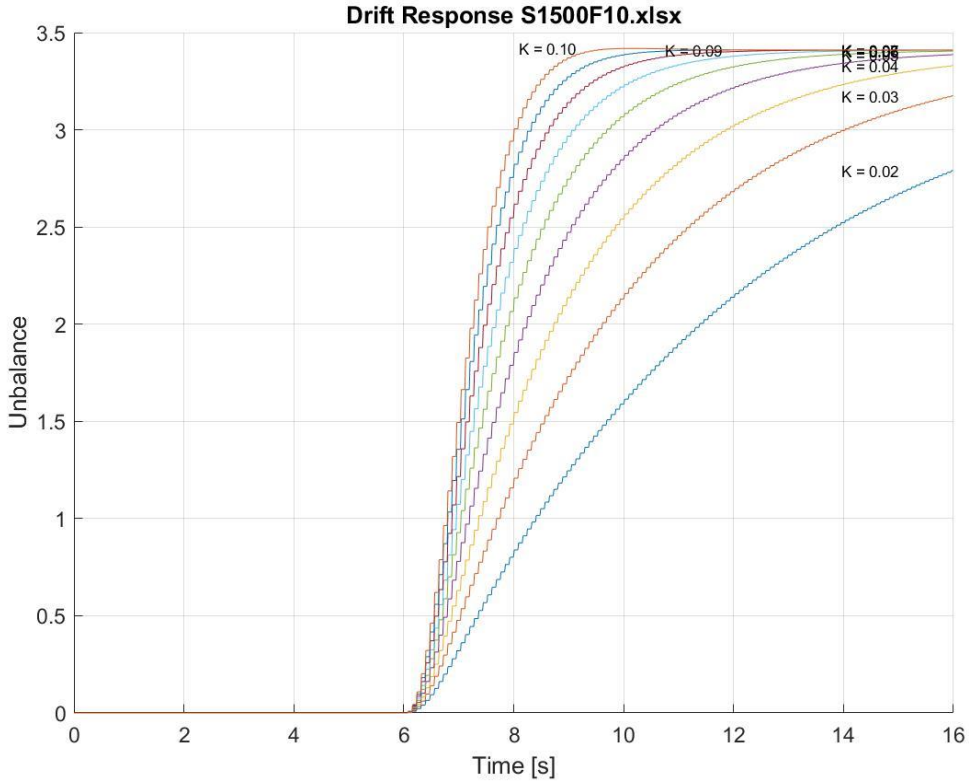


Figure 4.5.8: Integral Gain Simulation.

In Figure 4.5.9 are shown 4 different engine points that were tested for the current calibration. As it is possible to notice from the map that was calibrated with the former procedure, in Figure 4.5.10, the results are perfectly comparable and obtained with 1/3 of the total amount of the tests that were previously performed.

For example considering the point defined by 10 mm^3 and 1500 rpm (S1500F10 in the figure) the optimal integral gain calculated by the tool is $k = 0.11$ while the gain that was determined by means of testing in the former procedure is $k = 0.0996 \sim 0.1$. Instead, by looking at point 7 mm^3 x 700 rpm the integral gain calculated by the tool is $k = 0.04$ while the gain obtained by testing is $k = 0.0195 \sim 0.02$, one may be induced to think that the difference is substantial, but, it is important to highlight that the value listed in the INCA calibration is the results of trial and error tests which at some point must be interrupted since it is not feasible to iterate many times on one engine point due to costs and time limitations. This means that the value that was calibrated is a satisfactory value obtained in one of the trial and error tests but this does not mean that this value is the optimal one, instead, simulations at the laptop lasts fraction of second and can be iterated many times getting to the optimal value.

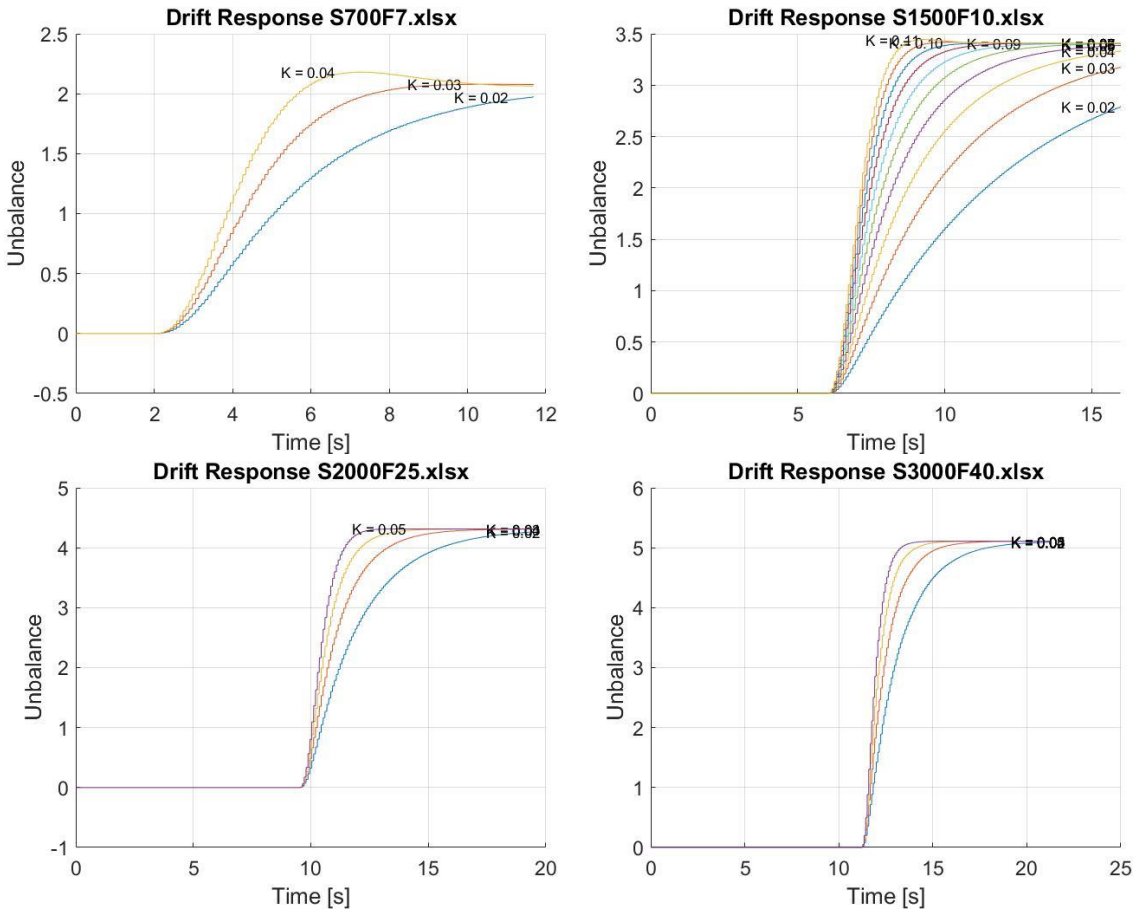


Figure 4.5.9: Tested points with current cal. procedure.

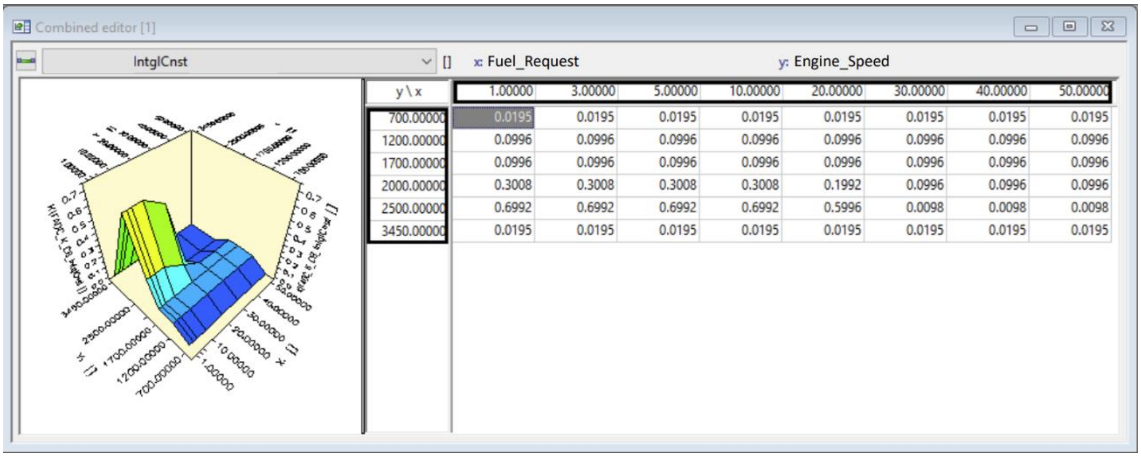


Figure 4.5.10: Former calibration of the gains.

4.6 Limits

CB Corrections have an upper and lower limit as defined in Chapter 3.3.4. Briefly, the lower limit is set as to avoid misfire while the upper limit is set in order to avoid damage to the engine.

To determine the upper limit there are different paths that can be followed. One of them provides to have available the IFI map, which is the map that allows to simulate what is the type of drift on an injector when it is around its end of life and the Injector map.

By getting the IFI map relative to the worst case acceptable $(+/-3\sigma)$ Table 5, at a given rail pressure and ET basic one can extrapolate what could be the actual injected quantity when an injector is aged. For example, suppose to run an engine with new injectors in idle mode and the rail pressure is 50 MPa and the fuel request is 3 mm^3, from Table 1 I retrieve the ET basic which is 301 μs. The ET basic is the energizing time that is wanted on the injector with its physical characteristics that are the nominal ones.

Due to aging the physical characteristics of the injector vary and the quantity that is injected with an energizing time equal to 301 μs at 50 MPa is not the expected one. To simulate this quantity mismatch on a new injector the IFI map is built, it makes available many ΔET values that are added to the ET basic in order to simulate the quantity mismatch on a new injector.

Table 5: Example of IFI map.

ET/MPa	**30**	**50**	**70**	**90**	**110**	**130**	**150**
150	-31,4	-31,1	-30,6	-30,4	-29,7	-28,9	-28,2
200	-42,0	-41,6	-40,8	-40,0	-37,2	-34,9	-32,9
250	-52,4	-51,4	-48,4	-43,4	-39,0	-35,4	-33,1
300	-62,7	-58,0	-50,4	-40,6	-35,9	-35,0	-34,3
350	-70,8	-61,5	-48,7	-42,7	-41,6	-40,8	-39,6
400	-77,0	-62,6	-51,1	-47,3	-47,9	-46,6	-44,7
500	-85,6	-62,4	-53,7	-58,9	-60,7	-58,1	-54,4
600	-87,4	-58,9	-65,0	-71,8	-73,9	-69,6	-63,7
700	-83,2	-68,9	-76,6	-85,4	-87,5	-81,0	-72,3
800	-73,0	-79,0	-88,4	-99,2	-101,5	-92,3	-80,2

The IFI map in Table 5 is relative to the worst case acceptable, meaning that those types of drifts will occur on an injector that needs to be replaced soon. In order to calculate the upper limits for the CB corrections this is taken in account, indeed the upper limits are calculated as the fuel quantity that is needed to compensate the drift calculated through the 3σ IFI.

Getting back to the previous example, supposing to have an ET basic equal to 301 μs at 50 MPa, from the IFI map it is get that the drift, in terms of ET, corresponds to -58 μs (negative drift has to be compensated with positive correction) and the drifted ET is then 243 μs.

By getting back to the Injector Map it is possible to retrieve what is the fuel request at the same rail pressure with the drifted ET which corresponds to the drifted quantity injected, about 1.8 mm^3. In this way, by subtracting the drifted fuel request to the basic fuel request the correction needed to compensate such a drift is retrieved, the maximum correction then amounts to 3-1.8 = 1.2 mm^3.

This operation is repeated at each breakpoint of the limits map and it is then calibrated in this way. In order to make faster this process it was built a tool that takes as input the injector map and the IFI map, together with the desired breakpoints of the limits map, as output it gives back the full limits map.

5 Conclusions and Future Work

One of the main goals of this thesis work was to reduce the effort to calibrate the Cylinder Balance control system. Many tools were developed in order to reduce the human factor influence in calibrations, for example the first tool that was developed is for the smoothing of the conversion gains. This procedure was previously performed by hand on Excel trying to give a satisfying trend to the curves, with the tool the procedure is now automatized and based on mathematical theory, meaning that is reproducible and consistent while saving much time. Moreover, most of the tools aimed at reducing the effort at the test facility together with the time to calibrate, for example the tool for the cylinder assignment and the integral constants. In Table 6 the times needed to calibrate the control system adopting the former procedure and the times needed to calibrate the CB adopting the tools and the techniques developed in the new calibration procedure are listed.

Table 6: Time efforts for former and new Cal. procedure.

	Time at Desk [hr]		Time on Vehicle [hr]	
	Former Cal.	New Cal.	Former Cal.	New Cal.
CBE Mean Removal	0.5	0.5	0.5	0.5
Enabling Conditions	8	2	8	0
Conversion to ET and Assignment	4	2	10	3
Integral	4	2	16	3
Limits	2	0.5	4	4
Calibration Time	18	7	38	10.5

It is possible to notice from the table that a great amount of time is saved with respect to the former procedure, moreover, most of the time is saved from the test activities and this is a paramount important aspect, since the control system has to be calibrated at the roller test bench and performing tests at such a facility is very costly. Once the control system has been calibrated following the new calibration procedure, comparing the former calibration with the new is important to evaluate its effect on the control system.

In terms of enabling conditions there is not a real metrics on how evaluating their performances, since the control system is wanted to work in full range, once it is always activated their performances are met. Although, when the control system is activated the cylinder assignment, and the integral constants play a fundamental role, since they account respectively for which cylinder is corrected and what is the rate of change of the corrections.

Taking into account the cylinder assignment, it was calibrated in two situations. At first without the enabling of the CBE mean value removal (as it was for the former calibration of the CB), and in a second place with this feature enabled, this was done since the mean value removal affects the assignment. The first calibration was performed in order to have a comparison with the former procedure, and the result was that the calibration provided by the tool is exactly the same as the former calibration, moreover, it was obtained by saving 7 hours at the roller test bench.

The Integral constants calibration has always been one of the trickiest calibration, since it required at least 3 tests to be performed on vehicle at each breakpoint of the map, requiring a total of 16 hours at the roller test bench. What was previously performed in order to calibrate these labels, was to change by hand the integral constant at a specified engine point in order to evaluate the unbalances response once enabling the CB. The integral constant at that engine point was considered to be calibrated once the convergence of the unbalances reached the steady state value with acceptable overshoot in a reasonable time. The tool allowed to deliver a calibration by performing just one test at each engine point, acquiring the unbalance response to an injector drift and feeding the results to a Simulink with a simulated CB control system. It allowed to simulate the unbalance responses at a varying integral constant value without having to perform this operation by hand on the vehicle. The results obtained with the tool are comparable to the ones obtained by hand on vehicle in the former calibration. Differences in the calibrations may be due to the fact that in the former procedure, due to timing reasons, an integral constant giving a good performance, but not an optimal one, was considered to be good

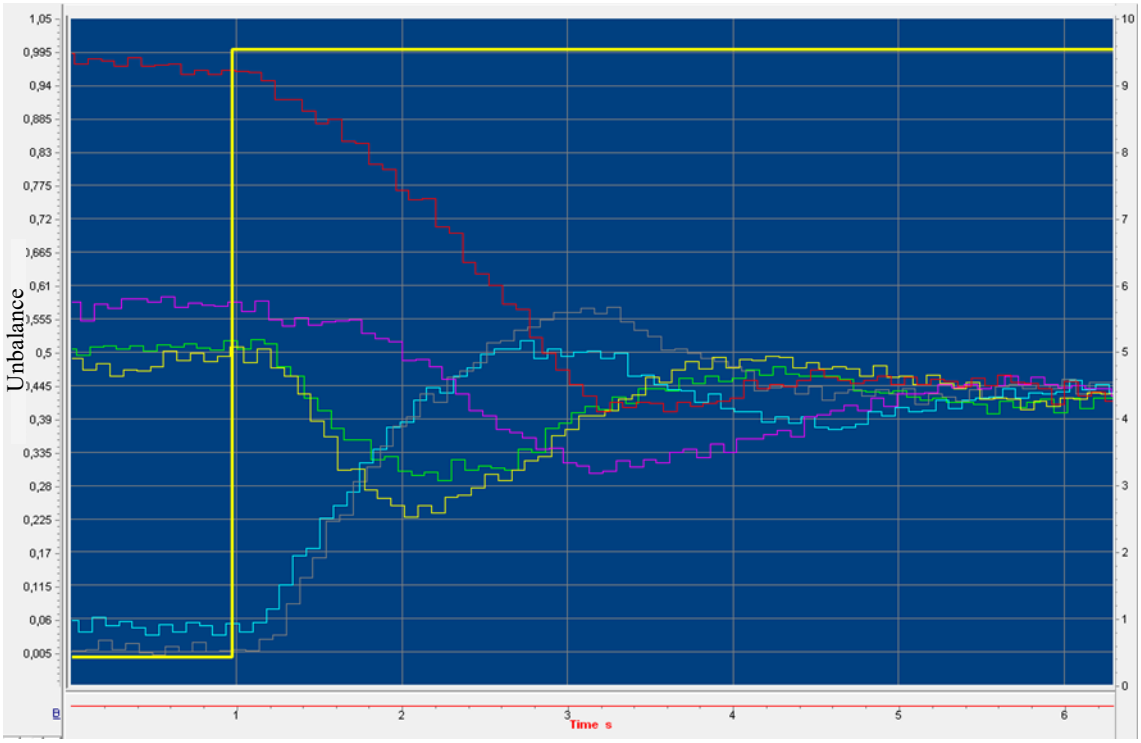


Figure 4.6.1: Speed 2000, Fuel Request 25, IntglCnst 0.1

for the purpose. For example, by looking at the response in Figure 4.6.1, the integral constant was calibrated at that engine point equal to $K = 0.1$, for the sake of calibration it was considered a satisfying unbalance response at CB enabling, although, some oscillations are still present. The tool optimizes with many different integral constants until the oscillations are not present anymore, indeed at that engine point the tool calculates an integral constant equal to 0.05 which is lower than the one shown on vehicle, meaning that it is most likely to eliminate the oscillations at that engine point. This is the explanation why some differences may be found in the integral constants calibrated through the tool with respect to the ones calibrated with the former procedure.

For other calibrations the process was automated, meaning that the results are the same delivered by the former procedure but obtained through the use of a script or tool, as it happens for examples for the limits map which allowed to save 1.5 hours at desk.

For what it concerns the Dead-Band calibration, an activity on IMEP dispersion was begun. This aimed at understanding which is the correlation between the IMEP dispersion, not measurable in real time on vehicle, and the engine orders, which instead are fed to the CB control system and represent an indirect measure of IMEP dispersion. Ongoing testing activities showed that IMEP dispersion behavior could be correlated with the amplitude of the engine orders that are overlapped to the primary crank wheel signal. If confirmed by ongoing testing activities, a possible calibration procedure could be calibrating the Dead-Band value as the value of unbalance that corresponds to the situation in which the IMEP dispersion is negligible. Some activities concerning the calibration of the CB during Cranking were started as well, and a possible calibration that was thought of is to calibrate the CB in order to neglect the Cranking event and activate it in normal mode when this event ends.

The new calibration procedure brought a huge step forward in terms of time effort reduction during calibration, while keeping the performances of the previously calibrated system. A further step that can be made in order to bring to almost zero the time effort for calibration, is to build a complete inertial model of the engine and driveline in order to be able to deliver calibrations without the need of the test facility for a first calibration delivery but only for validation purposes.

6 Codes

6.1 $K_{Conversion}$ Map

In this chapter all the codes that compose the tool for the calculation of the Gains Map are grouped.

6.1.1 Main

```
from cProfile import label
import pandas as pd
import numpy as np
import math
from matplotlib import pyplot as plt
from Linear_Smoothing_Utils import smoother

################## Take a look at the Read_Me file #################

WINDOW_SIZE = 5
THRESHOLD = 1.5
FILE_NAME_INPUT = "Excel_InjTbl/Injector_Map.xlsx"
FILE_NAME_OUTPUT = 'Excel_InjTbl/KtFULC_K_ET_CB_InjTblSlope_w5.xlsx'


def main():

    data_smoother = smoother(
        FILE_NAME_INPUT, FILE_NAME_OUTPUT, WINDOW_SIZE, THRESHOLD)
    # This object gives me back the excel file with the gains calculated
    data_smoother.smooth()
    data_smoother.grapher()

    ######### Here it is plotted the trend of the energizing times ET ############

    Smoothed_InjTblM = np.array(pd.read_excel(FILE_NAME_OUTPUT))

    Resultant_ET = Smoothed_InjTblM

    for j in range(1, Smoothed_InjTblM.shape[1]):

        for i in range(0, Smoothed_InjTblM.shape[0]):

            Resultant_ET[i, j] = Smoothed_InjTblM[i, 0]*Smoothed_InjTblM[i, j]

```

```
    plt.figure()

    for j in range(1, Resultant_ET.shape[1]):
        plt.plot(Resultant_ET[:, 0], Resultant_ET[:, j])

    plt.title('Fuel_Request*K_Conv @ Constant P --> ET')
    plt.grid("minor")
    plt.xlim([0, max(Resultant_ET[:, 0])-10])
    plt.xlabel('Fuel Request [mm^3]')
    plt.ylabel('K_Conv')
    plt.show()


if __name__ == "__main__":
    main()
```

6.1.2 Utils

```
from cProfile import label
import pandas as pd
import numpy as np
import math
from matplotlib import pyplot as plt


class smoother():

    def __init__(self, file_name_input, file_name_output, window=7, threshold=1.2):

        self.file_name_input = file_name_input
        self.file_name_output = file_name_output
        self.window = window
        self.threshold = threshold
        self.Inj_Table = []
        self.Inj_TableM = []

    def extract_data(self):

        # The Injector map is uploaded and the InjectorM map is initialized
        self.Inj_Table = np.array(pd.read_excel(self.file_name_input))
        self.Inj_TableM = np.zeros(
            [self.Inj_Table.shape[0]-1, (self.Inj_Table.shape)[1]-1])

```

```
        ########################First rough calculation of the InjectorM map as : (ET(2)-ET(1))/((Qty(2)-Qty(1)))#########################

        for j in range(1, (self.Inj_Table.shape)[1]):

            for i in range(1, (self.Inj_Table.shape)[0]-1):

                self.Inj_TableM[i-1, j-1] = (self.Inj_Table[i+1, j]-self.Inj_Table[i, j])/(
                    self.Inj_Table[i+1, 0]-self.Inj_Table[i, 0])

        # Plotting results before smoothing
        for j in range(0, (self.Inj_TableM.shape)[1]):

            plt.plot(self.Inj_Table[1:, 0], self.Inj_TableM[0:, j],
                     marker="^", label=f"Pressure {self.Inj_Table[0,j+1]}")

    def smooth_data(self):
        ########################################SMOOTHENING WITH LINEAR INTERPOLATION#########################################################

        self.Inj_TableM_Smooth = self.Inj_TableM  # Initializing the smoothed matrix

        # In this section the coefficient of the line are calculated within the self.window of points, after that, if the value of the function is below
        # the line it is set with the value of the line in that point, otherwise, if the value of the function is above the line it is left
        # as it is, except for the case in which it is above the line of a quantity equal to self.threshold* value(of the line)

        for j in range(0, (self.Inj_TableM_Smooth.shape)[1]):

            for i in range(0, (self.Inj_TableM_Smooth.shape)[0]-math.ceil(self.window/2)):

                if i < math.floor(self.window/2):

                    # x self.window of the fit
                    x_fit = self.Inj_Table[1:1+self.window, 0]
                    # y self.window of the fit
                    y_fit = self.Inj_TableM_Smooth[0:self.window, j]

                    fit = np.polynomial.polynomial.polyfit(x_fit, y_fit, 1)
                    value = np.polynomial.polynomial.polyval(
                        self.Inj_Table[i+1, 0], fit)

                elif i > ((self.Inj_TableM_Smooth.shape)[0]-math.ceil(self.window/2)):

                    x_fit = self.Inj_Table[((self.Inj_TableM_Smooth.shape)[
```

```
                                    0]-
math.ceil(self.window/2)), (self.Inj_TableM_Smooth.shape)[0], 0]  # x self.window of the fit
                y_fit = self.Inj_TableM[((self.Inj_TableM_Smooth.shape)[
                                     0]-
math.ceil(self.window/2)), (self.Inj_TableM_Smooth.shape)[0], j]  # y self.window of the fit

                fit = np.polynomial.polynomial.polyfit(x_fit, y_fit, 1)
                value = np.polynomial.polynomial.polyval(
                    self.Inj_Table[i+1, 0], fit)

            else:

                x_fit = self.Inj_Table[i+1-math.floor(
                    self.window/2):i+1+math.ceil(self.window/2), 0]
                y_fit = self.Inj_TableM_Smooth[i-math.floor(
                    self.window/2):i+math.ceil(self.window/2), j]

                fit = np.polynomial.polynomial.polyfit(x_fit, y_fit, 1)
                value = np.polynomial.polynomial.polyval(
                    self.Inj_Table[i+1, 0], fit)
                """
                if fit[1]>0:
                    fit[1]=0
                    print(fit[1])
                """

            if self.Inj_TableM_Smooth[i, j] > self.threshold*value or self.Inj_TableM[i, j] < value:

                self.Inj_TableM_Smooth[i, j] = value

    ############################################################Final Cleaning#####
    ############################################################

    def finalize_smoothing(self):
        # Adjusting for the values that create local minimum in the Map, the logic implemented is: st
arting from the end of the vector
        # Check if the current value is bigger than the following one, if it is, change the next value wit
h the current one
        for j in range(0, (self.Inj_TableM_Smooth.shape)[1]):

            for i in range(0, (self.Inj_TableM_Smooth.shape)[0]):

                if i > 0 and self.Inj_TableM_Smooth[(self.Inj_TableM_Smooth.shape)[0]-
i, j] > self.Inj_TableM_Smooth[(self.Inj_TableM_Smooth.shape)[0]-i-1, j]:

                    self.Inj_TableM_Smooth[(self.Inj_TableM_Smooth.shape)[
                        0]-i-1, j] = self.Inj_TableM_Smooth[(self.Inj_TableM_Smooth.shape)[0]-i, j]

```

```
110.        # In this section the pressure breakpoint is adjusted following the following logic: Imagine ha
    ving a fuel req of 50mm^3 it must not be allowed
111.        # thata 40 bar pressure line is higher than a 30 bar pressure line at same fuel req
112.
113.        for i in range(0, (self.Inj_TableM_Smooth.shape)[0]):
114.
115.            for j in range(0, (self.Inj_TableM_Smooth.shape)[1]-1):
116.
117.                if self.Inj_TableM_Smooth[i, j] < self.Inj_TableM_Smooth[i, j+1]:
118.
119.                    self.Inj_TableM_Smooth[i, j +
120.                                           1] = self.Inj_TableM_Smooth[i, j]
121.
122.        # Plotting the smoothed function
123.        for j in range(0, self.Inj_TableM.shape[1]):
124.
125.            plt.plot(self.Inj_Table[1:, 0], self.Inj_TableM_Smooth[0:, j],
126.                     '--', label=f"serie {j} smooth")
127.
128.    def write_data(self):
129.        self.Inj_TableM_write = pd.DataFrame(
130.            self.Inj_TableM_Smooth, columns=[self.Inj_Table[0, 1:]])
131.        self.Inj_Table_mm3 = pd.DataFrame(
132.            self.Inj_Table[1:self.Inj_Table.shape[0], 0], columns=["mm^3"])
133.        self.Inj_Table_Merged = pd.concat(
134.            [self.Inj_Table_mm3, self.Inj_TableM_write], axis=1, join="inner")
135.        self.Inj_Table_Merged.to_excel(self.file_name_output, index=0)
136.
137.    def smooth(self):
138.
139.        self.extract_data()
140.        self.smooth_data()
141.        self.finalize_smoothing()
142.        self.write_data()
143.
144.    def grapher(self):
145.
146.        plt.title("Conversion Table to ET")
147.        plt.ylabel("K_Conv")
148.        plt.xlabel("Fuel Request [mm^3]")
149.        plt.grid("minor")
150.        plt.show()
151.
```

6.2 CAWA

In this section the codes developed for the crank angular window assignment tools are grouped.

6.2.1 Main WAP

```
from cProfile import label
from re import A
import pandas as pd
import numpy as np
import matplotlib as plt
import sys
from CAW_Evaluation_Utils import CAW_Evaluator
import os
##################################----------------------MAKE SURE-------------------------
#####################################

# 1) Make sure you are not working with the tool inside a folder that is synched with OneDrive etc
. this could cause probs

# 2) Make sure you change the name of the ouput files (Change FILENAME_OUTPUT) inbetween d
ifferent analysis otherwise the
# tool would try to overwrite the content of an already existing excel file in the Results folder resul
ting into an error

# 3) Just select the parameters you prefer and fill the initialization section below, onche your data
in excel format are
# inside of folder Acquisitions you can launch the script

##################################----------------INITIALIZATION SECTION----------------
#######################################

# This is the cylinder i have drifted range from A to H (8 Cylinders maximum)
UNBAL_CYL = "A"
# TYPE_OF_DRIFT --- May be either "Positive" or "Negative"
TYPE_OF_DRIFT = "Positive"
# NUMBER_OF_CYLINDERS --- Number of cylinder in the engine
NUMBER_OF_CYLINDERS = 6
# MAX_SPEED_OVERRUNN --- Maximum speed at which the CBE is active
MAX_SPEED_OVERRUNN = 3500
LOWER_FUEL_THRESHOLD = 2
# TOLERANCE --- If the drifted cylinder is not the highest/lowest point but still in this range from
# the highest/lowest point then its assignment is considered valid. This allows to clean the results
TOLERANCE = 0.1
DEAD_BAND = 0.5
# GR_SHIFT_NEGLECTION --
- This is a boolean and is set either True or Flase depending if it is wanted that a window of points
# is neglected during gearshifting, this allows to clean the results
```

```
GR_SHIFT_NEGLECTION = False
####################################----------OUTPUT MANAGEMENT----------######################################
# The name following Acquisitions\Results\..... is the name of the excel file given in ouput containing a sheet for each test
FILENAME_OUTPUT = 'Acquisitions/Results/Res_neglected={}_GrShift_Tol_{}.xlsx'.format(
    GR_SHIFT_NEGLECTION, TOLERANCE)

####################################----------ANALYSIS----------######################################

DIRECTORY = 'Acquisitions'


def main():

    # This section calculated the WAP (Wrongly assigned points) by neglecting the time instants when a gearshift occurs since
    # GR_SHIFT_NEGLECTION is set True
    if GR_SHIFT_NEGLECTION == True:

        # Iterating through all the tests excel files in the folder "Acquisitions"
        for filename in os.listdir(DIRECTORY):

            name, extension = os.path.splitext(filename)
            # this avoids having porblems in wiritng the sheet name whihc has a max of 31 characters
            name = name[:18]

            FILE_NAME_INPUT = os.path.join(DIRECTORY, filename)

            if extension == '.xlsx':  # Checks if the file is an excel file

                print('Working on {}'.format(name))
                Evaluation = CAW_Evaluator(
                    FILE_NAME_INPUT, FILENAME_OUTPUT, UNBAL_CYL, TYPE_OF_DRIFT, NUMBER_OF_CYLINDERS, TOLERANCE, MAX_SPEED_OVERRUNN, LOWER_FUEL_THRESHOLD, DEAD_BAND, name)
                Evaluation.assignement_evaluation_neglected_grshift()
                Evaluation.export_wr_ass_points()
        Evaluation.plotter()

# This section calculated the WAP (Wrongly assigned points) by considering the time instants when the gearshift occurs
    else:

        for filename in os.listdir(DIRECTORY):

            name, extension = os.path.splitext(filename)
```

```
            # this avoids having porblems in wiritng the sheet name which has a max of 31 characters
            name = name[:18]

            FILE_NAME_INPUT = os.path.join(DIRECTORY, filename)

            if extension == '.xlsx':
                print('Working on {}'.format(name))
                Evaluation = CAW_Evaluator(
                    FILE_NAME_INPUT, FILENAME_OUTPUT, UNBAL_CYL, TYPE_OF_DRIFT, NUMBER_OF_CYLINDERS, TOLERANCE, MAX_SPEED_OVERRUNN, LOWER_FUEL_THRESHOLD, DEAD_BAND, name)
                Evaluation.assignement_evaluation_with_grshift()
                Evaluation.export_wr_ass_points()

        Evaluation.plotter()


if __name__ == "__main__":
    main()
```

6.2.2 Main Frequency at WAP

```
from cProfile import label
from re import A
from typing import Type
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import font_manager as fm
import sys
from CAW_Evaluation_Utils import CAW_Evaluator
import os


FUEL_WINDOW_LOW = 0
FUEL_WINDOW_HIGH = 25

SPEED_WINDOW_LOW = 1200
SPEED_WINDOW_HIGH = 1600

TIME_WINDOW_LOW = 0
TIME_WINDOW_HIGH = 200000000000

FILENAME = 'Acquisitions\Results\Res_neglected=False_GrShift_Tol_0.1.xlsx'
CYLINDER_NUMBER = 6
####--------------------------------------PROCESS--------------------------------------####
FIGURE_INITIAL_NAME = 'ALL_TESTS_Assign_Freq_Sp_{}_{}_Fu_{}_{}'.format(
    SPEED_WINDOW_LOW, SPEED_WINDOW_HIGH, FUEL_WINDOW_LOW, FUEL_WINDOW_HIGH)
```

```
27.
28.
29. def main():
30.
31.     excel = pd.ExcelFile(FILENAME)
32.     sheets = excel.sheet_names
33.     test_n = 0
34.     # This is a vector containing the frequencies of assignment for each cylinder for example
35.     # Frequency_Assigned_Cylinder_vec[0]--
    > Contains the number of times cylinder A was assigned
36.     # whenever the assignment is wrong (red dot) for all the tests
37.     Frequency_Assigned_Cylinders_vec = np.zeros(CYLINDER_NUMBER)
38.     # This is a matrix containing in each row the number of times each cylinder was assigned worng
    ,
39.     # each row is a single test,the sum of the columns is the total number of times the cylinder was
40.     # assigned during all the tests
41.     Frequency_Assigned_Cylinders_mat = np.zeros(
42.         [len(sheets), 8])
43.
44.     ########### ----------Calculating the frequency during all tests----------
    ########
45.     for name_of_sheet in sheets:
46.
47.         # Data Import
48.         Data = pd.read_excel(FILENAME, sheet_name=name_of_sheet)
49.         Time_WAP = Data['WAP Times [s]']
50.         Fuel_req_WAP = Data['WAP Fuel Request [mm^3]']
51.         Speed_WAP = Data['WAP Speed [rpm]']
52.         Assigned_Cylinders = Data['Assigned Cylinder']
53.
54.         # Initializing the variables containing the number of assignments
55.         Cyl_A = 0
56.         Cyl_B = 0
57.         Cyl_C = 0
58.         Cyl_D = 0
59.         Cyl_E = 0
60.         Cyl_F = 0
61.         Cyl_G = 0
62.         Cyl_H = 0
63.
64.         # Counting for each tests what cylinder is assigned when the assignment is wrong
65.         for i in range(0, Time_WAP.shape[0]):
66.
67.             if SPEED_WINDOW_HIGH > Speed_WAP[i] > SPEED_WINDOW_LOW and FUEL_WINDOW_
    HIGH > Fuel_req_WAP[i] > FUEL_WINDOW_LOW and TIME_WINDOW_HIGH > Time_WAP[i] > TIM
    E_WINDOW_LOW:
68.
69.                 if Assigned_Cylinders[i] == 0 and CYLINDER_NUMBER > 0:
```

```
                Cyl_A += 1
            elif Assigned_Cylinders[i] == 1 and CYLINDER_NUMBER > 1:
                Cyl_B += 1
            elif Assigned_Cylinders[i] == 2 and CYLINDER_NUMBER > 2:
                Cyl_C += 1
            elif Assigned_Cylinders[i] == 3 and CYLINDER_NUMBER > 3:
                Cyl_D += 1
            elif Assigned_Cylinders[i] == 4 and CYLINDER_NUMBER > 4:
                Cyl_E += 1
            elif Assigned_Cylinders[i] == 5 and CYLINDER_NUMBER > 5:
                Cyl_F += 1
            elif Assigned_Cylinders[i] == 6 and CYLINDER_NUMBER > 6:
                Cyl_G += 1
            elif Assigned_Cylinders[i] == 7 and CYLINDER_NUMBER > 7:
                Cyl_H += 1

        Frequency_Assigned_Cylinders_mat[test_n, :] = [
            Cyl_A, Cyl_B, Cyl_C, Cyl_D, Cyl_E, Cyl_F, Cyl_G, Cyl_H]
        test_n += 1

    ####----------Summing all tests----------####
    for i in range(0, CYLINDER_NUMBER):

        Frequency_Assigned_Cylinders_vec[i] = np.sum(
            Frequency_Assigned_Cylinders_mat[:, i])

    # Cylinder names
    Cylinders = [0, 1, 2, 3, 4, 5, 6, 7]
    Cylinders_letters = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

    bar_plot = plt.bar(Cylinders[0:CYLINDER_NUMBER], Frequency_Assigned_Cylinders_vec[0:CYLIN
DER_NUMBER],
                       tick_label=Frequency_Assigned_Cylinders_vec[0:CYLINDER_NUMBER])

    # Initializing the percentage of assignment & Calculating the rate of assignemnt
    Percentage = [0, 0, 0, 0, 0, 0, 0, 0]

    for i in range(0, CYLINDER_NUMBER):

        Percentage[i] = (Frequency_Assigned_Cylinders_vec[i] /
                         (np.sum(Frequency_Assigned_Cylinders_vec[0:CYLINDER_NUMBER])))*100

    # This part is only used to plot the labels on top of each bar inside the barplot
    i = 0
    for rect in bar_plot:

        height = rect.get_height()
        plt.text(rect.get_x() + rect.get_width() / 2.0, height,
```

```
                '{} ({}%)'.format(int(height), float(f'{Percentage[i]:.1f}')), ha='center', va='bottom', font='T
    imes New Roman', fontsize=11)
        i += 1

    plt.rcParams["font.family"] = "serif"
    plt.rcParams["font.serif"] = ["Times New Roman"]
    plt.title('Assignment at WAP',
              font='Times New Roman', fontsize=14)
    plt.xlabel('Cylinder', font='Times New Roman', fontsize=14)
    plt.ylabel('n° of Assignments',
               font='Times New Roman', fontsize=14)
    plt.xticks(
        Cylinders[0:CYLINDER_NUMBER], Cylinders_letters[0:CYLINDER_NUMBER], font='Times New
    Roman')
    plt.yticks(font='Times New Roman')

    name, extension = os.path.splitext(FILENAME)

    plt.savefig('Acquisitions\Results\Bargraphs\Bar_{}'.format(
        FIGURE_INITIAL_NAME))
    plt.show()


if __name__ == "__main__":
    main()

```

6.2.3 Utils

```
from cProfile import label
from operator import index
from statistics import mode
from matplotlib.pyplot import plot
from matplotlib.transforms import Transform
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import sys
import xlsxwriter
import openpyxl as pxl
from os.path import exists
import os.path
import time


```

```
17. class CAW_Evaluator():
18.
19.     def __init__(self, file_name_input, file_name_output, Unbal_Cylinder_letter="Z", Drift_Type="Not
    Specified", Cyl_Number=-1, Tolerance=0, Overrun_Speed=-
    1, Lower_Fuel_Thrsh=0, Dead_Band=0, Test_Name='No Name'):
20.
21.         self.file_name_input = file_name_input
22.         self.Overrun_Speed = Overrun_Speed
23.         self.file_name_output = file_name_output
24.         self.Unbal_Cylinder_letter = Unbal_Cylinder_letter
25.         self.Drift_Type = Drift_Type
26.         self.Cyl_Number = Cyl_Number
27.         self.Tolerance = Tolerance
28.         self.Lower_Fuel_Thrsh = Lower_Fuel_Thrsh
29.         self.Dead_Band = Dead_Band
30.         self.Test_Name = Test_Name
31.         self.Unbalances = []
32.         self.Time = []
33.         self.Fuel_Req = []
34.         self.Speed = []
35.         self.Gear = []
36.         self.wrongly_assigned_fuel = []
37.         self.wrongly_assigned_speed = []
38.         self.wrongly_assigned_time = []
39.         self.Unbal_Cylinder = []
40.         self.Assigned_Cylinder = []
41.         self.GrShiftFlag = []
42.
43.     def check_inputs(self):
44.
45.         # Here the unbalanced cylinder is converted from letter to number in order to be used efficie
    ntly in the code
46.         if self.Overrun_Speed < 0 or self.Overrun_Speed > 10000:
47.             sys.exit("\n\nTHE OVERRUN SPEED HAS NOT BEEN SET CORRECTLY\n\n")
48.         elif self.Unbal_Cylinder_letter == "A" or self.Unbal_Cylinder_letter == "a":
49.
50.             self.Unbal_Cylinder = 0
51.         elif self.Unbal_Cylinder_letter == "B" or self.Unbal_Cylinder_letter == "b":
52.
53.             self.Unbal_Cylinder = 1
54.         elif self.Unbal_Cylinder_letter == "C" or self.Unbal_Cylinder_letter == "c":
55.
56.             self.Unbal_Cylinder = 2
57.         elif self.Unbal_Cylinder_letter == "D" or self.Unbal_Cylinder_letter == "d":
58.
59.             self.Unbal_Cylinder = 3
60.         elif self.Unbal_Cylinder_letter == "E" or self.Unbal_Cylinder_letter == "e":
61.
```

```
62.             self.Unbal_Cylinder = 4
63.         elif self.Unbal_Cylinder_letter == "F" or self.Unbal_Cylinder_letter == "f":
64.
65.             self.Unbal_Cylinder = 5
66.         elif self.Unbal_Cylinder_letter == "G" or self.Unbal_Cylinder_letter == "g":
67.
68.             self.Unbal_Cylinder = 6
69.         elif self.Unbal_Cylinder_letter == "H" or self.Unbal_Cylinder_letter == "h":
70.
71.             self.Unbal_Cylinder = 7
72.         else:
73.             sys.exit("\n\nCHECK IF YOU HAVE ASSIGNED THE UNBALANCED CYLINDER CORRECTLY\n\
    nRemember that the tool works up to 8 cylinders engines and cylinders are name from A to H")
74.
75.         if self.Unbal_Cylinder > (self.Cyl_Number-1):
76.
77.             sys.exit("\n\n!!! THE UNBALANCED CYLINDER / NUMBER OF CYLINDERS SELECTED IS NOT
    CONSISTENT !!!\n\nRemember the tool works with up to 8 cylinders engines and they are called fr
    om A to H, the unbalanced cylinder must be between this range\n\n")
78.
79.     def extract(self):
80.
81.         self.check_inputs()
82.
83.         Data = pd.read_excel(self.file_name_input)
84.
85.         self.Time = Data['time']
86.         self.Speed = Data['YeEPSI_n_Lores']
87.         self.Fuel_Req = Data['VeFULC_V_FuelReq']
88.         try:
89.             self.Gear = Data['VeFADC_e_CB_TransGr']
90.         except:
91.             print("\n NO GEAR INFO \n")
92.
93.         self.Unbalances = np.zeros([self.Time.shape[0], self.Cyl_Number])
94.         for i in range(0, self.Cyl_Number):
95.             self.Unbalances[:,
96.                             i] = Data['VaFADC_V_CB_CylUnBalEstd[x]_[{}]'.format(i)]
97.
98.         # Is initialized here to be able to use it even when we want to neglect the gearshift
99.         self.GrShiftFlag = np.zeros(Data.shape[0])
100.
101.    def gr_shift_neglection(self):
102.
103.        self.neglection_window = 60
104.        # Here i create a new vector in which i put a flag n points before a gearshift (n = self.neglecti
    on_window)
```

```
105.        # this will be used later to skip the from the i point to the next i+self.neglection_window poin
    t
106.
107.        for i in range(1, self.Time.shape[0]):
108.
109.            if self.Gear[i] != self.Gear[i-1]:
110.
111.                if i > self.neglection_window:
112.
113.                    self.GrShiftFlag[i-self.neglection_window] = 1
114.                else:
115.
116.                    # if the gearshift occurs a number of samples lower than the window the
117.                    self.GrShiftFlag[0] = 1
118.                    # first sample is set as gearsshift in order to start to neglect the points since i am alre
    ady
119.                    # in a gearshift situation
120.
121.    def unbalance_consistency_check(self):
122.
123.        stepper = 0  # This variable is used to be able to jump steps when neglecting gearshifts
124.
125.        ################## POSITIVELY DRIFTED INJECTORS #############
126.
127.        if self.Drift_Type == "Positive" or self.Drift_Type == "positive":
128.
129.            ind = 0  # This is the index of the vectors containing all the wrongli assigned points listed
    below
130.
131.            wf = np.zeros(self.Unbalances.shape[0])
132.            ws = np.zeros(self.Unbalances.shape[0])
133.            wt = np.zeros(self.Unbalances.shape[0])
134.            ass_cyl = np.ones(self.Unbalances.shape[0])*self.Unbal_Cylinder
135.
136.            for i in range(0, self.Unbalances.shape[0]):
137.
138.                # Minimum unbalance in the time instant
139.                minimum_value = np.min(self.Unbalances[i, :])
140.                max_abs = np.max(np.abs(self.Unbalances[i, :]))
141.                ####-------------GEARSHIFTING----------####
142.
143.                if self.GrShiftFlag[i] == 1:
144.
145.                    stepper = i + self.neglection_window
146.
147.                ####-------------IT IS ENTERED IF THE STEPPER HAS NOT BEEN SET (NO Gr SHIFT)------
    -----####
148.
```

```
149.             elif i >= stepper:
150.
151.                 if self.Speed[i] < self.Overrun_Speed and self.Fuel_Req[i] > self.Lower_Fuel_Thrsh and
    abs(self.Unbalances[i, self.Unbal_Cylinder]) > self.Dead_Band:
152.
153.                     ####------------THE EVALUATION OF THE UNBALANCE IS PERFORMED HERE------
    ----####
154.
155.                     if self.Unbalances[i, self.Unbal_Cylinder] >= 0:
156.
157.                         # 1)For sure when drifting positively i expect the drifted cylinder to have a negati
    ve unbalance
158.                         # 2)What i am doing here is to neglect the points in which the unbalances are all
     positive-->CutOff
159.                         # 3)If the unbalance of the drifted cylinder is positive either is a bad assignemen
    t or cutoff
160.
161.                         if minimum_value < 0:
162.
163.                             # If the minimum unbalance is negative it means i am not in cutoff so it is a b
    ad assignement
164.                             # and has to be taken into account otherwise no action is taken
165.
166.                             index_minimum = np.where(
167.                                 self.Unbalances[i, :] == minimum_value)  # Index of the minimum unbalanc
    e corresponding to the cyl n
168.
169.                             ass_cyl[ind] = index_minimum[0][0]
170.                             wf[ind] = self.Fuel_Req[i]
171.                             ws[ind] = self.Speed[i]
172.                             wt[ind] = self.Time[i]
173.                             ind += 1
174.
175.                     elif self.Unbalances[i, self.Unbal_Cylinder] > (minimum_value + abs(self.Tolerance*
    max_abs)):
176.
177.                         # 1) When the unbalance of the drifted cylinder is negative it has to be checked i
    f is the lowest one
178.                         # 2) If the unbalance of the drifted cylinder is not the lowest one (with tolerance
    of self.Decision_Band)
179.                         # a bad assignement is present and the assigned cylinder is noted down
180.
181.                         index_minimum = np.where(
182.                             self.Unbalances[i, :] == minimum_value)
183.
184.                         ass_cyl[ind] = index_minimum[0][0]
185.                         wf[ind] = self.Fuel_Req[i]
186.                         ws[ind] = self.Speed[i]
```

```
187.                     wt[ind] = self.Time[i]
188.                     ind += 1
189.
190.         if ind > 0:
191.             self.wrongly_assigned_fuel = wf[0:ind-1]
192.             self.wrongly_assigned_speed = ws[0:ind-1]
193.             self.wrongly_assigned_time = wt[0:ind-1]
194.             self.Assigned_Cylinder = ass_cyl[0:ind-1]
195.
196.     ################## NEGATIVELY DRIFTED INJECTORS #############
197.
198.     if self.Drift_Type == "Negative" or self.Drift_Type == "negative":
199.
200.         print('\n\n\n\nIt has to be made specular to the other case\n\n\n\n')
201.
202.  def export_wr_ass_points(self):
203.
204.     # Here the data frame with the visited points (Speed/FuelReq) and Wrongly assigned points
    WAP (Speed/FuelReq/Assigned Cylinder) are constructed
205.     dfw = pd.DataFrame({'WAP Times [s]': self.wrongly_assigned_time, 'WAP Speed [rpm]': self.wr
ongly_assigned_speed,
206.                         'WAP Fuel Request [mm^3]': self.wrongly_assigned_fuel, 'Assigned Cylinder': self.
    Assigned_Cylinder})
207.     dfo = pd.DataFrame({'Time [s]': self.Time, 'Visited Speed [rpm]': self.Speed,
208.                         'Visited Fuel Request [mm^3]': self.Fuel_Req})
209.     dfmerged = pd.merge(dfo, dfw, left_index=True,
210.                         right_index=True, how='outer')
211.
212.     # If the excel file doesn't exists yet it is created and the first sheet is written
213.     if exists(self.file_name_output) == False:
214.
215.         dfmerged.to_excel(self.file_name_output, sheet_name='Test {}'.format(
216.             self.Test_Name), index=0)
217.         print('\nExcel_File has been created\n')
218.
219.     # If the excel file already exists a new sheet in the same file is created and written
220.     elif exists(self.file_name_output):
221.
222.         writer = pd.ExcelWriter(
223.             self.file_name_output, engine='openpyxl', mode='a')
224.         dfmerged.to_excel(writer, sheet_name='Test {}'.format(
225.             self.Test_Name), index=False)
226.         print('\nSheet -- {} -- has been created\n'.format(self.Test_Name))
227.         writer.save()
228.  # In the end i will have an excel file with a sheet for each test and in each sheet the visited point
    s and the wrongly assigned ones
229.     plt.plot(self.Speed, self.Fuel_Req, 'bo',
230.              alpha=0.1, zorder=1)
```

```
        plt.plot(self.wrongly_assigned_speed,
                 self.wrongly_assigned_fuel, 'ro', zorder=2)
        # Here is added the legend to each point to note down which is the assigned cylinder
        # for i in range(0, len(self.Assigned_Cylinder)):
        # plt.text(
        # self.wrongly_assigned_speed[i], self.wrongly_assigned_fuel[i], int(self.Assigned_Cylinder[i]), horizontalalignment='center', verticalalignment='center')

        plt.legend(['Visited Points', 'WAP'])

    def plotter(self):
        # This section allows to plot the figure when all the WAP (wrongly assigned points) were evaluated from all tests
        name, extension = os.path.splitext(self.file_name_output)
        plt.grid()
        plt.title('WAP During all tests')
        plt.xlabel('Engine Speed [rpm]')
        plt.ylabel('Fuel Request [mm^3]')
        plt.xlim([0, self.Overrun_Speed])
        plt.savefig('Acquisitions\Results\WAP-{}.png'.format(name[21:]))
        plt.show()

    def assignement_evaluation_with_grshift(self):

        self.extract()
        self.unbalance_consistency_check()
        time.sleep(5)  # Added to avoid problems in writing the excel files

    def assignement_evaluation_neglected_grshift(self):

        self.extract()
        self.gr_shift_neglection()
        self.unbalance_consistency_check()
        time.sleep(5)  # Added to avoid probelms in writing the excel files

```

6.3 Intgl_Cnsts Simulation

```
clc
clear all
close all
%%%%%%%%%%%%%%%%  INITIALIZATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this version i cycle through all the files and i cycle also through
% different values of integral constants

UNBALANCED_CYLINDER = 0;            %Unbalanced Cylinder 0=A,1=B,2=C,...
                                   %The drift maps VBasic + IFI for each
                                    %impulse enabled
```

```
FILTER_WINDOW = 20;                 %Moving average filter window for the unbalance
signal
POLES_NUMBER = 2;                   %Number of poles of the transfer funciton
ZEROS_NUMBER = 1;                   %Number of zeros of the transfer function

OVERSHOOT_THRESHOLD_DOWN = 3;       %Percentage Value
OVERSHOOT_THRESHOLD_UP = 4;         %Percentage Value
RISE_TIME_THRESHOLD = 1.7;          %Seconds
IFI_Rec_Percentage = 1.05;

%%%%%%%%%%%%%%% PROCESS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
path = ['C:\Users\pietr\OneDrive - Politecnico di
Torino\PUNCH\Development\CB_80_03\TF_Estimation_Matlab\Authomatized_Tool\Acquisitions
_Try1'];
fil = fullfile(path,'*.xlsx');
d = dir(fil);

ind = 0;

for k = 1:numel(d)

    ind = ind +1;%This is used to save tf in the struct
    %%%%%%%%%%%%%%  GETTING DATA FROM EXCEL  %%%%%%%%%%%%%
    filename = fullfile(path,d(k).name);
    Data = readtable(filename);
    Unbalance = Data.(['VaFADC_V_CB_CylUnBalEstd_x___', ...
        num2str(UNBALANCED_CYLINDER),'__XETK_1']);
    Time = Data.time- Data.time(1);
    IFI = Data.IFI;
    [FIELPATH,NAME,EXT] = fileparts(filename);
    DRIFT_mm3 = drift_pick(['C:\Users\pietr\OneDrive - Politecnico di
Torino\PUNCH\Development\CB_80_03\TF_Estimation_Matlab\Authomatized_Tool\Drifts'],NAM
E);%5.14; %[mm3]            %Equivalent fuel drift derived from sigma, calculated
throgu

    %%%%%%%%%%%%%%%%%%%%%% TF PARAMETERS  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%Understand the exat time when the IFI is turned on%%%%%%%%

    for i=2:length(Time)

        if abs(IFI(i)) > abs(min(IFI(IFI ~= 0))*IFI_Rec_Percentage)
            Samples_till_Step = i;
            break
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%% TF Input/Output  %%%%%%%%%%%%%%%%%%%%%%%%
    Input = DRIFT_mm3*[zeros(1,Samples_till_Step) ones(1,(length(Time)-
Samples_till_Step))]';
    Output = Unbalance;
    %%%%%%%%%%%%%%%%%% Estimating the transfer function  %%%%%%%%%%%%%%%%%
    [filename,name,ext] = fileparts(filename); % Used to save in a clear way

    Smoothed_Output = movavg(Output,'linear', FILTER_WINDOW);
```

```
    sample_time.(sprintf('Test_N%d',ind)) = (Time(100)-Time(99));
    TTime = zeros(length(Time),1);

    samp_sim = (sample_time.(sprintf('Test_N%d',ind)));
    Acq = iddata(Smoothed_Output, Input, sample_time.(sprintf('Test_N%d',ind)));

    H.(sprintf('Test_N%d', ind)) = tfest(Acq, POLES_NUMBER,
ZEROS_NUMBER,'Ts',sample_time.(sprintf('Test_N%d',ind)))
    DCGAIN.(sprintf('Test_N%d', ind)) = dcgain(H.(sprintf('Test_N%d', ind)));

    %%%%%% From here on different integral constants are simulated %%%%%%%%
    Overshoot = (OVERSHOOT_THRESHOLD_UP-OVERSHOOT_THRESHOLD_DOWN)/2 +
OVERSHOOT_THRESHOLD_DOWN;
    Undershoot = 0;
    Rise_Time = 1000;
    K_integral = 0.01;
    figure(ind)
    title(sprintf('Drift Response %s',d(k).name))
    xlabel('Time [s]')
    ylabel('Unbalance')
    set(gca,'FontSize',14)
    while ((Overshoot > OVERSHOOT_THRESHOLD_DOWN ||  Rise_Time > RISE_TIME_THRESHOLD)
&& Overshoot < OVERSHOOT_THRESHOLD_UP)

        K_integral = K_integral + 0.01
        %%%%%%%%%%% STarting the simulation of the transfer function %%%%%%%%%%
        Simout = sim('Integral_Constant_Simulation');
        [Overshoot ,Undershoot, Rise_Time] =
Overshoot_Estimation((Simout.Results.Time),(Simout.Results.Data),(DCGAIN.Test_N1*DRIF
T_mm3),500,K_integral);

    end

end
```

6.4 Limits

6.4.1 Main Limits

```
1.  from unicodedata import name
2.  import numpy as np
3.  import pandas as pd
4.
5.  from Limits_Calculation_Utils import interpolator
6.
7.  from decimal import InvalidContext
8.  from matplotlib.cbook import is_scalar_or_string
9.  import pandas as pd
10. import numpy as np
11.
```

```
12.
13. ####################----------INITIALIZING----------
    ############################
14.
15. filename = 'Limit_Calculation_Data.xlsx'
16.
17. # INJECTOR TABLE
18. InjTable = np.array(pd.read_excel(filename, sheet_name='Injector_Table'))
19. # IFI TABLE
20. IFITable = np.array(pd.read_excel(filename, sheet_name='IFIMap'))
21. # DESIRED BREAKPOINTS
22. # Pressure Bp
23. Desired_BpRailP = np.array(pd.read_excel(
24.     filename, sheet_name='BpRailPressure'))
25.
26. # Fuel_Req Bp
27. Desired_BpFuelReq = np.array(pd.read_excel(filename, sheet_name='BpFuelReq'))
28.
29.
30. def main():
31.     ############----------INJECTOR TABLE MANIPULATION ----------
    ##############
32.     Interpolator = interpolator()
33.     ############PRESSURE interpolation is performed on pressure breakpoints##############
    ########
34.     Pressure_Bp_Interpolation_InjTbl = Interpolator.Table_Column_interpolation(
35.         InjTable, Desired_BpRailP, InjTable[0, :])
36.
37.     # Below the resultant map interpolated on P Bp
38.     InjTbl_Interpolated_PBp = Interpolator.Column_Interp_Table
39.
40.     ##########################FUEL_REQ Interpolation on Fuel req breakpoints##########
    ############
41.     # Since the tool interpolates on column the result from the previous step must be transposed
42.     # and interpolated again in the same way
43.     Fuel_Rq_Bp_Interpolation_InjTbl = Interpolator.Table_Column_interpolation(
44.         np.transpose(InjTbl_Interpolated_PBp), Desired_BpFuelReq, InjTable[:, 0])
45.
46.     # Below the resultant map interpolated also on Fuel_Req Bp
47.     InjTable_New_Bp = np.transpose(Interpolator.Column_Interp_Table)
48.     print('\nInjector Table New Break Points')
49.     print(InjTable_New_Bp)
50.
51.     ############----------IFI MAP MANIPULATION----------
    ######################
52.
53.     Pressure_Bp_Interpolation_IFI = Interpolator.Table_Column_interpolation(
54.         IFITable, Desired_BpRailP, IFITable[0, :])
55.
56.     # Below the resultant map interpolated on Pressure Bp
57.     IFITable_Interpolated_PBp = Interpolator.Column_Interp_Table
58.
59.     # Creating a vector with the desired ET Breakpoints which correspond to all the ET in the
60.     # new Injector Table. It makes easier to find the correspondent drift at each ET_Basic
61.     # Initializing the vector of ET_Basic's Breakpoints
62.     ET_Desired_Bp = np.zeros(
```

```
        [1, (InjTable_New_Bp[1:, 1:].shape[0]*InjTable_New_Bp[1:, 1:].shape[1] + 1)])
    ET_Desired_Bp[0, 0] = 0
    ind = 0

    for ET_row in InjTable_New_Bp[1:, 1:]:
        for ET in ET_row:
            ET_Desired_Bp[0, ind] = ET
            ind += 1
    # Sorting in ascending order
    ET_Desired_Bp.sort()

    # Interpolating the IFI ET_Bp on the desired ones (Previously determined)
    ET_Bp_Interpolation_IFI = Interpolator.Table_Column_interpolation(
        np.transpose(IFITable_Interpolated_PBp), ET_Desired_Bp, IFITable[:, 0])

    # Below the IFI map with the wanted breakpoints
    IFITable_New_Bp = np.transpose(Interpolator.Column_Interp_Table)
    print('\nIFI Map New Break Points')
    print(IFITable_New_Bp)

    #####---------------------- CALCULATING THE DRIFTED ET INJECTOR MAP -----------
##############

    Drift_ET_Calculation = Interpolator.drifted_ET_map(
        InjTable_New_Bp, IFITable_New_Bp, max(IFITable[:, 0]))

    # Below the result of Injector Table + Drift at each ET_Basic
    Drifted_ET_InjTable = Interpolator.Drifted_ET_Map
    print('\nDrifted ET Injector Table')
    print(Drifted_ET_InjTable)

    #####----------------- CALCULATING THE DRIFTED QUANTITY INJECTOR MAP ----------------
######

    Drift_Quantity_Calculation = Interpolator.delta_drifted_quantity_map(
        Drifted_ET_InjTable, InjTbl_Interpolated_PBp)

    print('\nDrifted Quantity Table')
    print(Interpolator.Drifted_Qty_Table)
    dDrifted_Qty = Interpolator.Delta_Drifted_Qty_Table
    print('\nDelta (V_Basic - (V_Basic+Drift)) Drifted Quantity Table')
    print(dDrifted_Qty)

    ###############----------------CALCULATING THE LIMITS MAP-----------------------
##########

    Limits_Map_Calculation = Interpolator.Limits_map_calc(dDrifted_Qty)

    Limits_Map = Interpolator.limits_map

    print('\n LIMITS MAP')
    print(Limits_Map)

    # Creating the excel file with results
    df = pd.DataFrame(np.transpose(Limits_Map))
    df.to_excel('Results\Limits_Map.xlsx',
```

```
            sheet_name='Limits_Map', index=0)


if __name__ == '__main__':
    main()

```

6.4.2 Utils Limits

```
from decimal import InvalidContext
from matplotlib.cbook import is_scalar_or_string
import pandas as pd
import numpy as np
import sys


class interpolator():

    def __init__(self):
        pass
        """
        Table --> Injector Table

        Desired_Bp --> Are the desired breakpoints for the interpolated map

        Avlbl_Bp --> Are the breakpoints available in the original map

        Interpolation_Direction --
> Defines wether the map has to be interpolated by rows or by columns. (i.e. to interpolate a 2D
map one needs
        to do a first interpolation on columns and after that perform the same on rows or eitherway)
        """

    def extract_data(self, Table, Desired_Bp, Avlbl_Bp):

        self.Table = Table
        self.Desired_Bp = Desired_Bp
        self.Avlbl_Bp = Avlbl_Bp
        self.Column_Interp_Table = []
        self.Result = []

    def Table_Column_interpolation(self, Table, Desired_Bp, Avlbl_Bp):
        """
        Table --> Provide the table you would like to interpolate on new column breakpoints

        Desired_Bp --> Are the desired breakpoints that the interpolated map should have

        Avlbl_Bp --
> Are the breakpoints that the original map has which correspond to the first row of the table (Ta
ble[0,:])
        """

        self.extract_data(Table, Desired_Bp, Avlbl_Bp)

```

```
42.         col_index = 0
43.         self.Column_Interp_Table = np.zeros(
44.             [self.Table.shape[0], max(self.Desired_Bp.shape)])
45.
46.         for i in range(0, max(self.Desired_Bp.shape)):
47.
48.             # BREAKPOINT AVAILABLE
49.             # The zero is needed because when is recognized puts the breakpoints at index 0
50.             if self.Desired_Bp[0, i] in self.Avlbl_Bp:
51.
52.                 self.Column_Interp_Table[:, col_index] = self.Table[:, np.where(
53.                     self.Desired_Bp[0, i] == self.Avlbl_Bp)[0][0]]  # Finds the index of the breakpoint that c
    orresponds to the desired value and substitute the column
54.                 col_index += 1
55.
56.             # BREAKPOINT NOT AVAILABLE
57.             else:
58.
59.                 # Checks that the desired brekpoint is not at the boundary of the map
60.                 if self.Desired_Bp[0, i] < max(self.Avlbl_Bp) and self.Desired_Bp[0, i] > min([i for i in self.A
    vlbl_Bp if i > 0]):
61.
62.                     self.Column_Interp_Table[0,
63.                                              col_index] = self.Desired_Bp[0, i]  # Value of the breakpoint at index 0
64.                     right_col_index = np.where(
65.                         self.Desired_Bp[0, i] < self.Avlbl_Bp)[0][0]
66.
67.                     for j in range(1, self.Table.shape[0]):
68.
69.                         self.Column_Interp_Table[j, col_index] = ((self.Table[j, right_col_index]-
    self.Table[j, right_col_index-1])/(
70.                             self.Avlbl_Bp[right_col_index]-self.Avlbl_Bp[right_col_index-
    1]))*(self.Desired_Bp[0, i]-self.Avlbl_Bp[right_col_index-1])+self.Table[j, right_col_index-1]
71.
72.                 # if the desired breakpoint is less than the minimum Bp of the Table it is ramped to 0 si
    nce it is wupposed to be found at the lower bound of the map
73.                 elif self.Desired_Bp[0, i] < min([i for i in self.Avlbl_Bp if i > 0]):
74.
75.                     self.Column_Interp_Table[0,
76.                                              col_index] = self.Desired_Bp[0, i]
77.                     right_col_index = 1
78.
79.                     for j in range(1, self.Table.shape[0]):
80.
81.                         self.Column_Interp_Table[j, col_index] = ((self.Table[j, right_col_index]-0)/(
82.                             self.Avlbl_Bp[right_col_index]-0))*(self.Desired_Bp[0, i]-0)+0
83.
84.                 # If the Bp is greather than the largest is kept constant to the largest value
85.                 elif self.Desired_Bp[0, i] > max(self.Avlbl_Bp):
86.
87.                     self.Column_Interp_Table[:,
88.                                              col_index] = self.Table[:, (self.Table.shape[1]-1)]
89.                     self.Column_Interp_Table[0,
90.                                              col_index] = self.Desired_Bp[0, i]
91.
92.                 col_index += 1
```

```
93.
94.     def drifted_ET_map(self, Injector_Table, IFI_Table, Max_Et_OriginalIFI):
95.         """
96.         Injector_Table --
    > Provide the previously interpolated Injector Table on the new breakpoints (to obtain use: Table_
    Column_interpolation twice (rows and columns))
97.
98.         IFI_Table --
    > Provide the previously interpolated IFI Table on the new breakpoints (to obtain use: Table_Colu
    mn_interpolation twice (rows and columns))
99.         """
100.
101.        self.Injector_Table = Injector_Table
102.        self.IFI_Table = IFI_Table
103.        self.Drifted_ET_Map = np.zeros(self.Injector_Table.shape)
104.        self.Max_Et_OriginalIFI = Max_Et_OriginalIFI
105.
106.        self.Drifted_ET_Map[0, :] = self.Injector_Table[0, :]
107.        self.Drifted_ET_Map[:, 0] = self.Injector_Table[:, 0]
108.
109.        for j in range(1, self.Injector_Table.shape[1]):
110.
111.            for i in range(1, self.Injector_Table.shape[0]):
112.                # At this point i am sure that the IFI table has all the needed brekpoints since i have buil
    t it on the desired breakpoints
113.                if self.Injector_Table[i, j] > Max_Et_OriginalIFI:
114.                    self.Drifted_ET_Map[i, j] = 0
115.
116.                else:
117.                    drift = self.IFI_Table[(
118.                        np.where(self.Injector_Table[i, j] == self.IFI_Table[:, 0])[0][0]), j]
119.
120.                    self.Drifted_ET_Map[i,
121.                                        j] = self.Injector_Table[i, j] + drift
122.
123.    def delta_drifted_quantity_map(self, Drifted_ET_Table, Injector_Table):
124.        """
125.        Drifted_ET_Table --
    > It is the Drifted Injector Map on the new Breakpoints, it is then obtained by:
126.                            1)Interpolate the injector Map on the new BreakPoints (use: Table_Column_int
    erpolation twice (rows and columns))
127.                            2)Interpolate the IFI on the new BreakPoints          (use: Table_Column_interpol
    ation twice (rows and columns))
128.                            3)At each Injector Map Breakpoint add its drift       (use: drifted_ET_map)
129.                            4)The Drifted_ET_Table is obtained
130.
131.        Injector_Table  --
    > Provide the Injector table after interpolating it ONLY on the desired Pressure breakpoints, it is i
    mportant to leave the
132.        Fuel_Request breakpoints at their original values (use: Table_Column_interpolation once only
     on the desired pressure breakpoints)
133.        """
134.        # Coming from the previous section is the drifted injector map
135.        self.Drifted_ET_Table = Drifted_ET_Table
136.        # It is the injector map that was found on the new brekpoints
137.        self.Injector_Table = Injector_Table
```

```
138.         self.Drifted_Qty_Table = np.zeros(self.Drifted_ET_Table.shape)
139.         self.Drifted_Qty_Table[0, :] = self.Drifted_ET_Table[0, :]
140.         self.Drifted_Qty_Table[:, 0] = self.Drifted_ET_Table[:, 0]
141.         self.Delta_Drifted_Qty_Table = np.zeros(self.Drifted_ET_Table.shape)
142.         self.Delta_Drifted_Qty_Table[0, :] = self.Drifted_ET_Table[0, :]
143.         self.Delta_Drifted_Qty_Table[:, 0] = self.Drifted_ET_Table[:, 0]
144.
145.         for j in range(1, self.Drifted_Qty_Table.shape[1]):
146.
147.             for i in range(1, self.Drifted_Qty_Table.shape[0]):
148.
149.                 if self.Drifted_ET_Table[i, j] in self.Injector_Table[1:, j]:
150.
151.                     self.Drifted_Qty_Table[i, j] = self.Injector_Table[np.where(
152.                         self.Drifted_ET_Table[i, j] == self.Injector_Table[1:, j])[0][0], 0]
153.
154.                 else:
155.
156.                     if self.Drifted_ET_Table[i, j] < max(self.Injector_Table[1:, j]) and self.Drifted_ET_Table[i, j
    ] > min(self.Injector_Table[1:, j]):
157.
158.                         bottom_index = np.where(
159.                             self.Drifted_ET_Table[i, j] < self.Injector_Table[1:, j])[0][0] + 1  # one is added sinc
    e we search from row 1 on
160.
161.                         # Linear Interpolation
162.                         self.Drifted_Qty_Table[i, j] = ((self.Injector_Table[bottom_index, 0]-
    self.Injector_Table[bottom_index-1, 0])/(self.Injector_Table[bottom_index, j] -
163.                                                 self.Injector_Table[bottom_index-
    1, j]))*(self.Drifted_ET_Table[i, j]-self.Injector_Table[bottom_index-
    1, j])+self.Injector_Table[bottom_index-1, 0]
164.
165.                     elif self.Drifted_ET_Table[i, j] < min(self.Injector_Table[1:, j]):
166.
167.                         bottom_index = 1
168.                         wanted_x = self.Drifted_ET_Table[i, j]
169.                         known_x = [
170.                             self.Injector_Table[bottom_index, j], 0]
171.                         known_y = [
172.                             self.Injector_Table[bottom_index, 0], 0]
173.                         self.Drifted_Qty_Table[i, j] = ((self.Injector_Table[bottom_index, 0]-0)/(
174.                             self.Injector_Table[bottom_index, j] - 0))*(self.Drifted_ET_Table[i, j]-0)+0
175.
176.                     elif self.Drifted_ET_Table[i, j] > max(self.Injector_Table[1:, j]):
177.
178.                         self.Drifted_Qty_Table[i,
179.                                                j] = self.Injector_Table[i, 0]
180.
181.         for j in range(1, self.Drifted_ET_Table.shape[1]):
182.
183.             for i in range(1, self.Drifted_ET_Table.shape[0]):
184.
185.                 self.Delta_Drifted_Qty_Table[i, j] = -self.Drifted_Qty_Table[i,
186.                                                      j] + self.Drifted_Qty_Table[i, 0]
187.
188.     def Limits_map_calc(self, Delta_Quantity_Tbl):
```

```
189.
190.        self.limits_map = Delta_Quantity_Tbl
191.
192.        for i in range(1, Delta_Quantity_Tbl.shape[0]):
193.
194.            for j in range(1, Delta_Quantity_Tbl.shape[1]):
195.
196.                if Delta_Quantity_Tbl[i, j] == Delta_Quantity_Tbl[i, 0]:
197.
198.                    self.limits_map[i, j] = 0
199.
200.                else:
201.
202.                    self.limits_map[i, j] = Delta_Quantity_Tbl[i,
203.                                                               0] + Delta_Quantity_Tbl[i, j]
204.
```

7 Bibliography

[1] Pietro Sansone, Mario Lavella, “Friction dissipation in internal combustion engines: cam tappet contact”. In: IOP Conf.Ser.:Mater. Sci. Eng. 1038 012014

[2] S.H Gawande, L.G. Navale, M.R. Nandgaonkar, “Power Balancing if Inline Multicylinder Diesel Engine”, In: Hindawi Publishing Corporation, Volume 2012, ID 937917

[3] Magdi K. Khair, Hannu Jaaskelainen, “Diesel Fuel Injection”, In: DieselNet

[4] Hannu Jaaskelainen, Magdi K. Khair, “Fuel Injection System Components”, In: DieselNet

[5] Raul Payri, Pedro Martì-Aldavarì, Tomas Montiel, Alberto Viera, “Influence of aging of diesel injector on multiple injection strategies”, In: Applied Thermal Engineering, vol. 181, no. August, p. 115891, 2020. DOI: 10.1016/j.applthermaleng.2020.115891

[6] O. Hofmann, P. Strauß, S. Schuckert, B. Huber, D. Rixen, G. Wachtmeister, “Identification of Aging Effects in Common Rail Diesel Injectors Using Geometric Classifiers and Neural Networks”, in: SAE Technical Papers 2016- April (2016). doi:10.4271/2016-01-0813.