POLITECNICO DI TORINO

Department Of Mechanical and Aerospace Engineering

Master's Degree Course

in Automotive Engineering – Management of Industrial Processes

Master's Degree Thesis

Development of a prototype of an ECU in Arduino environment for didactical purposes

Relatore

prof. Paolo Crovetti

Candidato

Arianna Iraldo

Academic year 2021/2022

Table of Contents

Table of Figures

1. Introduction

The main purpose of this dissertation is that of providing an educational tool which may be useful in the understanding of the working principles of an Electronic Control Unit for Internal Combustion Engines, and may constitute a tutorial medium used in the course of "Electronic Systems for Vehicles" at Politecnico di Torino.
The course covers a wide range of subjects, giving an extensive perspective over on-board systems, starting from an overview of microprocessors and microcontrollers working principles, up to the explanation of communication protocols, sensors and actuators, engine control modes and Electronic Control Unit design flow. The willing of integrate the already widespread program comes from the first-hand experience as a student: even if clear explanations of theoretical principles are at the base of a proper understanding of the subject, engagement and ability to remember what has been discussed exponentially increase when students have the possibility to concretize their knowledge. Being Electronics a subject which requires a wide range of theorical prerequisites to be fully embraced, a practical tutorial which gives engineers-to-be the chance to experience and try a tangible tool may result in enrichment of an already extensive program.
The state of art of Electronic Systems courses in different Automotive Engineering universities has been analysed and will be discussed in following chapter, so that to highlight the current methods adopted by teaching staff all around the world: specifically, an overview will be proposed over the most important and renowned universities providing Automotive Engineering courses. Apart from exceptional cases, where students are given the possibility to deepen their knowledge by means of lab experiences in this context, it appears that almost all Automotive Courses are completely devoted to mechanics and the teaching of electric and electronic systems is usually powertrain-oriented. A deep understanding of Electronics principles is generally out of the scope of an Automotive Engineering student. The purpose of this dissertation is to propose a possible solution to this lack, i.e., a prototype of an Electronic Control Unit, which has been designed in order to be used during course tutorials. The hope is that students, having the possibility to handle a control unit, to appreciate its basic architecture and its software implementation, and being able to modify the code, the wirings and the inputs, may increase their skills, visualise theorical notions and better understand course topics.
To this end, the unit has been implemented - based on Arduino platform and MATLAB sensor

simulation - with a main focus over the adoption of a simplified methodology, which may result in a straightforward understanding of the functionalities an actual ECU implements in regard to engine states control.

The choice of an Arduino board in the design process seemed optimal from the start: being Arduino an open-source platform, consisting of a programmable hardware (microcontroller) and a software (IDE, i.e., Integrated Development Environment), it results to be itself a powerful didactic tool, thanks to its user-friendly interface, its easy-to-comprehend workbook and its toolkit, inclusive of everything needed for a beginner usage. This platform has been introduced to implement a simplified ECU working principles, while main sensors used in the engine control (rpm sensor, coolant temperature sensor, temperature sensor of EGO sensor) have been virtualized using MATLAB code. On the other hand, in order to better show serial communication and to make the platform more engaging, the TPS sensor (i.e., Throttle Position Sensor) has been implemented by a potentiometer assembled on Arduino board: by changing its angular position it is possible to switch from an engine state to the other, of course based also on data gathered from other sensors. The assembling of the potentiometer, together with a slide-switch on the board - simulating the ON-OFF engine states - may result in the acquisition of some notions regarding hardware architecture and wirings, also giving the possibility to students to improve the unit by adding other components, by following the same methodologies. Microphones, LEDs and many other components present in Arduino tool-kit may be assembled taking the provided code as a reference. In this regard, also from software point of view, code deployment and explanation may be useful to better understand how to implement a control algorithm, to acquire knowledge regarding C and C++ languages and to be introduced to serial communication principles.

The thesis will start from the previously mentioned overview over methods and tools adopted by main Automotive Engineering courses, which will take place in Chapter 2: the focus in this phase is put over the general approaches adopted by educational programs in dealing with Electronics; possible lacks or suggestions will be highlighted, in order to depict a general scenario in which to place Politecnico di Torino state of art.

Chapter 3, on the other hand, will describe the theory at the base of the ECU control activity, together with the methodologies adopted during prototype design flow: the main goal of this chapter is to outline the reasons why engine control is needed, together with a description of the control algorithm implemented by the ECU, the role every sensor occupies in control activities, their working principles and vehicle electronic architecture. An explanation regarding simulation conditions and simplification level will be also provided in this chapter.

Chapter 4 will concern the Arduino hardware architecture: specifically, analog and digital pins working principles will be described, together with electric connections and wirings; circuit connecting potentiometer and slide-switch will be discussed and depicted. A brief overview over software principles and serial monitor will also be provided.
Chapter 5 will explain the specific functions and logic adopted during code drafting: serial communication principles, analog inputs, functions used in MATLAB to simulate the behaviour of the sensors and plots will be depicted and described.

The last chapter will gather conclusions and lessons learnt during the prototype design and the drafting of this thesis.

2. Didactics overview

An overview over didactics in different parts of the world is presented in this chapter, in order to make a comparison between different educational realities: this results to be a beneficial activity, which permits to place Politecnico di Torino courses into a general scenario and eventually to get ideas from other approaches.
During the drafting of this essay, it resulted clear that Automotive Engineering courses all around the world are mainly devoted to the different branches of mechanics and to manufacturing processes: CAD engineering, fluid mechanics, science of materials, thermodynamics, structural analysis and manufacturing technologies are usually the major subjects discussed during both Bachelor and Master of Science programs (apart, of course, from mathematics and algebra). Nevertheless, it is sufficient to analyse the last decades to understand the extent to which electronics and electric systems gained importance, also regarding vehicles embedded with thermal engines: up to the Seventies, electronic systems were mainly used to increase power, while efficiency became the main goal of Automotive and Mechanics Engineers starting from early 1980s. Nevertheless, a modern engine is designed so that to achieve maximum power with minimum effort in terms of fuel consumption: this is achieved by means of electronic control of ignition timing, fuel injection timing, valve opening, sensors and actuators. Nowadays, the goal of a proper vehicle design is also put on safety: advanced driver-assistance systems (ADAS) are now of capital importance, to the point that General Safety Regulation of the European Union states that new commercial vehicles need to possess six ADAS features (Moving-Off Information System, Blind Spot Information System, Reversing Information System, Intelligent Speed Assist, Driver Drowsiness & Alertness Warning and Tire Pressure Monitoring System), starting from mid-2022 for homologations and mid-2024 for registrations. Each of these functionalities is implemented by one or even more than one ECUs, which communicate with each others by means of different communication protocols (LIN, CANbus, FlexRay, etc.) in order to increase driving safety.
Beside from power-oriented topics and safety concerns, electronics can make a capital difference also for what concerns the comfortability of the vehicle: an efficient network, advanced infotainment system and a proper human-machine interface may make a manufacturer achieve a great advantage with respect to their competitors.
For these reasons, it seems clear that the educational preparation of Automotive engineers should

not be limited just to that physics branches strictly related to mechanical design; it is extremely important to achieve a proper acknowledgment also for what concerns all those skills which permit an engineer to be up to date.
Unfortunately, even if every Automotive Engineering career usually offers Electric and/or Electronic Systems course, it is clear that a wide discussion is provided mainly to students attending Electric Engineering, Machine Learning and similar careers. Moreover, very often, students do not have the possibility to gain a first-hand experience of components and microprocessors. This could result in a not proper understanding of the subject and may cause a lack of interest in the topic.
In the following pages, Table 1 will be proposed, reporting a didactic state of art of the first 20 universities, ranked on the basis of Automotive Engineering courses[1]. Politecnico di Torino, which has been listed on the 16th place in this list, is here neglected.

Missing information are due to a difficulty in retrieving detailed course descriptions and curricula.

UNIVERSITY	**Ranking**	**Department of Automotive Engineering**	**Electronics credits**	**First-hand experience**
Tsinghua University [2]	1	Present	Mandatory	N/A
University of Michigan - Ann Arbor [3]	2	Present	Mandatory, but only related to power and control systems	N/A
University of Wisconsin - Madison [4]	3	Not present. Courses are erogated by Mechanical Engineering Dept.	Courses list not available	N/A

Shanghai Jiao Tong University [5]	4	Not present. Courses are erogated by Mechanical Engineering Dept.	Courses list not available, but National Engineering Laboratory for Automotive Electronic Control Technology was set up.	Laboratory activities are related to system performance testing and validation, powertrain system level and vehicle level control system development, hardware-in-loop simulation, control strategy development and verification, and calibration,
Massachusetts Institute of Technology [6]	5	Not present. Courses are erogated by Mechanical Engineering Dept.	Courses list not available, but Sloan Automotive Laboratory was set up.	Laboratory research activities are related to engine efficiency and fuel consumption optimization, not strictly to electronics.
Beijing Institute of Technology [7]	6	Present.	Courses list not available	Vehicular Engineering department collaborate with Electromechanical Group in two National Engineering Practical Education Centers, but list of activities is not available.

Ohio State University [8]	7	Not present. Courses are erogated by Mechanical Engineering Dept.	Mandatory, but related to power electronics devices.	N/A
Aalborg University [9]	8	Automotive Engineering is not present; but a focus on automotive components is present in Mechanical Engineering program.	Mandatory, specifically the program involves an Embedded Micro Processors: Applications and C Programming course.	3rd semester involves the possibility to do a project-oriented stay in Denmark or abroad, where students can take part in the day-to-day operations in the business.
University of California - Berkeley [10]	9	Automotive Engineering course is not provided, but Mechanical Engineering students may choose elective courses such as Vehicle Dynamics and Control to specialize their curriculum.	As an elective course students may choose "Introduction to MEMS (Microelectromechanical Systems)"	N/A

Tongji University [11]	10	Present	Courses list not available.	N/A
Jilin University [12]	11	Present	Courses list not available.	N/A
RWTH Aachen University [13]	12	Present	A proper Electronics course is not present. The Master involves courses such as: Electric Drives and Storage Systems (mandatory course) Automated and Connected Driving Challenges (elective course) Control Engineering (elective course) Automated Driving (elective course)	Automated and Connected Driving Challenges involves the possibility to carry on a research project.
Tianjin University [14]	13	Present	Courses list is not available	N/A
Southwest Jiaotong University [15]	14	Not present. Courses are erogated by Mechanical Engineering Dept.	Courses list is not available	N/A

Texas A&M University - College Station [16]	15	Automotive Engineering course is not provided, but Mechanical Engineering students may choose elective courses to specialize their curriculum.	Mandatory course related to electronic controls.	N/A
Chalmers University of Technology [17]	17	Present. Two different curricula may be choosen: Automotive Engineering and Mobility Engineering	Mandatory courses are power-oriented only.	N/A

Delft University of Technology [18]	18	Automotive Engineering course is not provided, but Mechanical Engineering students may choose specialisation programs such as: Perseption and Modelling (camera, radar, lidar) Dynamics and Control Human Factors (HMI)	Courses list not available	N/A
University of Tokyo [19]	19	Automotive Engineering course is not provided, but Mechanical Engineering students may choose elective courses to specialize their curriculum.	Mandatory course: Advanced MEMS and Microsystems	N/A

Virginia Polytechnic Institute and State University [20]	20	Automotive Engineering course is not provided, but Mechanical Engineering students may choose Automotive Major to specialize their curriculum.	Mandatory courses: Vehicle Control; Introduction to Programming in C; Mechatronics: Theory and Application.	Mechatronics course include laboratory experiences regarding software development, micro-controller technology and control applications.

Table 1 – Automotive Engineer didactics overview

This short overview among 20th best rated universities for what concerns Automotive Engineering reveals a general lack in terms of Electronics didactics: what can be pointed out is not only the necessity to adopt a practical approach which may enhance students' understanding of the subject, but, more important, to establish a wide and compulsory study of Electronics as a part of careers mechanics-oriented also. This could result into training Automotive Engineers with a wider perspective and more up-to-date with respect to today's market needs.

3. Methodology: Electronics fundamentals

The project is intended to provide students that first-hand experience whose lack has been pointed out in most of Automotive Engineering universities: this involves a simplified recreation of control activities performed by an ECU, taking as a reference a Spark Ignition (SI) internal combustion engine, equipped with a throttling device.

This chapter will mainly focus on a description of that Automotive Electronics fundamentals whose understanding is necessary to fully embrace the logic of this thesis.

In a conventional SI engine, the fuel must be vaporized and well mixed with the air inducted through the intake valve into the cylinder and then compressed; under normal operating conditions, combustion is initiated towards the end of the compression stroke at the spark plug by an electric discharge. High values of flame propagation speed can be achieved only if the air/fuel mixture is quite close to the stoichiometric ratio: for this reason, when an SI engine has to be operated at part load, it is not possible to reduce only the fuel while maintaining unchanged the air mass into the cylinder, but it is necessary to adopt a throttling device at the intake (throttle valve), while on the contrary, for CI engines, there are no strict requirements in terms of air/fuel ratio and the load can be varied by varying the amount of injected fuel per cycle. The TPS (i.e. Throttle Position Sensor) is a sensor used to monitor the throttle valve position and it is usually located on the throttle valve spindle; ignition timing and fuel injection timing are altered depending on the position of the throttle valve, and also depending on the rate of change of that position: because of that, this parameter is essential for the ECU to correctly implement its control functions, together with data gathered from rpm sensor, cooling medium temperature sensor, EGO (Exhaust Gas Oxygen sensor) temperature sensor and mass-air-flow sensor (MAF sensor).

This chapter is intended to explain the theory at the base of the ECU control activity, starting from vehicle electronic architecture, up to the analysis of the algorithm devoted to engine states control, and to a basic description of the working principles of the sensors, whose data are gathered in order to switch from an engine state to the other.

3.1. Vehicle electronic architecture

A high-level description of the Engine Control Unit should be provided, before approaching the dissertation regarding single sensors in Chapter 3.3, to properly clarify which are the signals necessary for the ECU to implement its control algorithm and where sensors and actuators are physically located.

As touched upon in Chapter 3.1, ECU switches from one state to another on the basis of information it retrieves from different sensors: TPS, MAF and Temperature sensors.

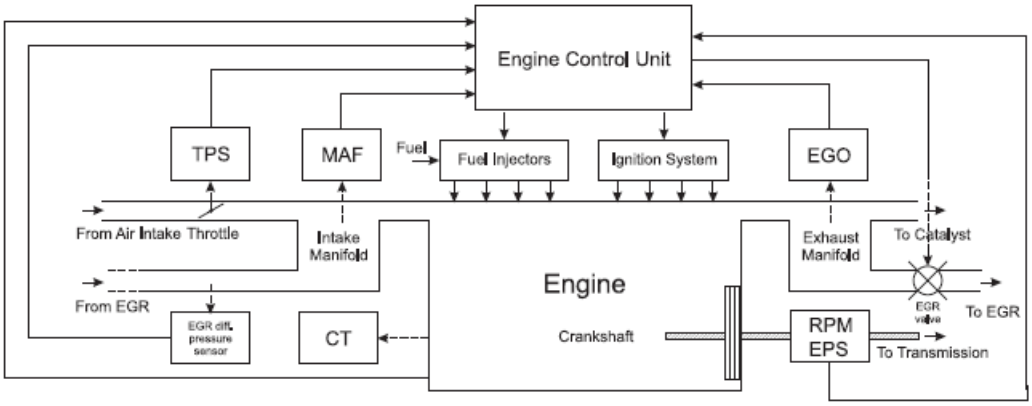

Figure 1 - Vehicle electronic architecture [21]

As depicted in Figure 1 - Vehicle electronic architecture, TPS sensor is placed in correspondence of the throttle valve, between the intake manifold and the air filter: it registers the information related to the throttle valve, which is electronically connected with accelerator pedal, and supplies and regulates the air flow required to form the air-fuel mixture. Thus, the stability of the engine operating modes, the level of fuel consumption and the characteristics of the car as a whole depend on the correct operation of the damper. In practical terms, in the open position, the pressure in the intake system is equal to atmospheric (if we are considering a naturally-aspirated engine); as it closes, the pressure decreases, approaching the vacuum value. Modern types of dampers involve an electrically operated and electronically controlled throttle, which means that there is not a mechanical interaction between the accelerator pedal and the damper, but instead an electronic control is used, which also allows the engine torque to be varied without the need to depress the pedal and the idle speed of the engine is automatically adjusted by moving the throttle. The

electronic control system also takes into account of the signals from the gearbox, climate control system, brake pedal position sensor and, when enabled, cruise control. The signal related to throttle valve position, retrieved by TPS, is necessary to the ECU to understand what is the position of the accelerator pedal, i.e., what is the power request by the driver in a specific moment; this enables the ECU to switch from Closed-Loop Mode to eventually hard acceleration or deceleration modes.

Mass-air-flow sensor, on the other hand, is positioned in the intake manifold and is necessary to the ECU in order to balance and deliver the correct fuel mass to the engine. Specifically, MAF signal is used by the Engine Control Unit to assess the fuel injection timing in Open-Loop mode, as described in Chapter 3.2.1. This signal provides a measured air flow information, while the oxygen sensor provides the closed-loop feedback in order to make minor correction to the predicted air mass.

RPM sensor is located on engine crankshaft and the signal is sent to the ECU so that it can monitor pistons speed and switch from one control mode to the other.

Temperature sensors, on the other hand, are located in different places in the vehicle: EGO temperature sensor, of course is located in correspondence of oxygen sensor, to verify whether the operating temperature of EGO sensor is reached or not (and eventually enabling the ECU to switch from Open-Loop mode to Closed-Loop mode), while coolant temperature sensor, in most cars, is installed near the thermostat in the cylinder head or block, or directly in the thermostat housing. A second coolant temperature sensor could be also installed in the radiator. A coolant temperature sensor (CTS, also known as ECT or ECTS sensor) is used to measure the temperature of the coolant/antifreeze mix in the cooling system, giving an indication of how much heat the engine is giving off. The sensor sends signals to the ECU, continually monitoring the cooling medium temperature to make sure the engine is running at the optimum temperature.

Oxygen sensors are usually two and are located in different places in the vehicle, but always inside the exhaust gas flow: most cars have one EGO sensor close to the engine, typically in the exhaust manifold and the second one generally is installed behind the catalytic converter. This helps to monitor catalyst performance by comparing the before and after readings.

As previously discussed, the goal of the Engine Control Unit is that of injecting the proper amount of fuel so that to achieve the correct air/fuel mixture and consequently the desired power and pollutant emissions outputs. For this reason, the signal related to correct injection timing, both that evaluated during Open-Loop mode and that corrected during Closed-Loop mode, is sent to

two different actuators: fuel injectors and spark plugs. This is due to the fact that optimizing combustion process involves both fuel injector timing by means of the ECU, valve opening (which can be controlled by means of the camshaft or electronically) and by the spark advance. A complete discussion regarding actuators working principles and combustion process optimization is out of the scope of this dissertation.

In the following chapters, ECU control modes and the actual working principles of the sensors which has been simulated during this project will be described.

3.2. ECU control modes

A robust combustion process is essential for smooth and reliable engine operation: it must be fast (it must occupy a small fraction of the total cycle time), so that the engine energy conversion process is efficient, and highly repeatable, so that variations from one cycle to the next are small. As depicted in Figure 2, the burning velocity peaks take place for slightly rich mixtures with respect to stoichiometric.

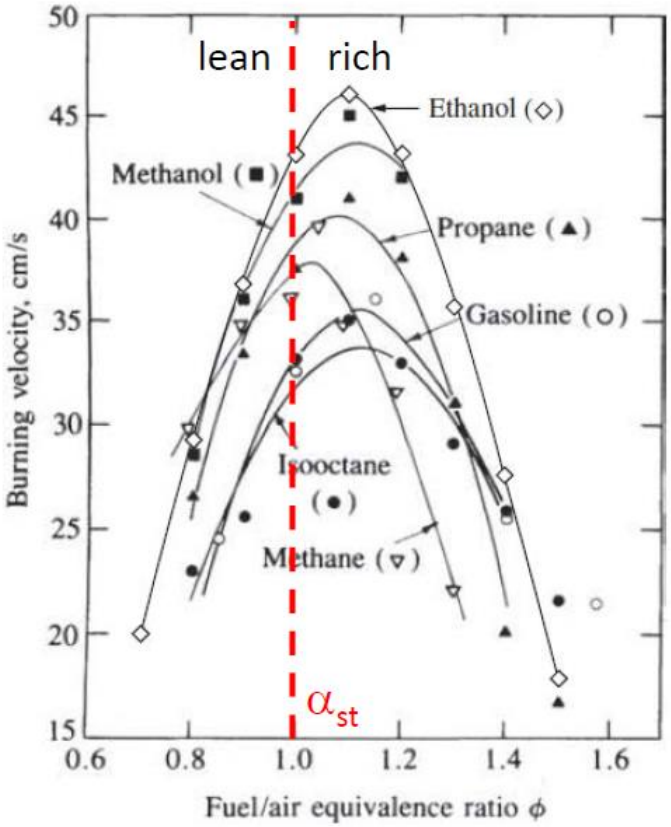

Figure 2 – Burning velocity for different fuels with respect to fuel/air equivalence ratio [22]

Engine control unit main purpose is that of evaluating in every operating condition the optimal driving parameters (i.e., amount of fuel delivered to cylinders, spark advance, exhaust gas recirculation, etc.) in order to achieve the maximum power. However, what must be taken into consideration, is that also achieving minimum pollutant emissions and reducing fuel consumption must be targets for an efficient engine control. These goals are actually achieved mainly by acting on air/fuel ratio, being it strictly related to brake mean effective pressure (bmep), combustion efficiencies and pollutants emission.

Figure 3 shows an example of a SI engine map, which summarizes a possible fuel metering system requirement: as depicted, maximum bmep could be achieved by enriching the mixture.

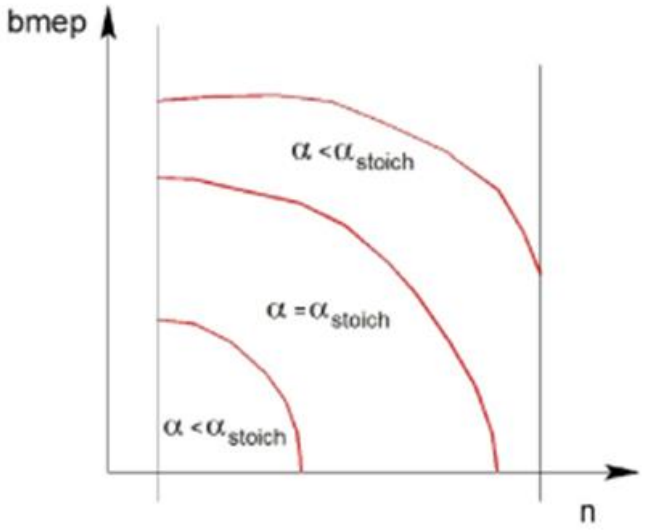

Figure 3 - bmep of lean, stoichiometric and rich mixtures with respect to rpm [23]

It is clear that the main purpose of the ECU for an efficient combustion process is maximizing torque, and then bmep, at full load, while at part-load fuel consumption has to be minimized. This is achieved by keeping the mixture quite close to stochiometric ratio while in nominal condition, and by leaving this state whenever high-power demand is provided (slightly rich mixture is injected) or on the contrary when the request of fuel is low, as in the case of a deceleration (in this case, slightly lean mixture is injected).

Another important aspect of the engine control is pollutants emission: as depicted in Figure 4, it can be noticed that for lean mixtures (which can be injected at part load) the engine produces lower HC and CO emissions, but NOx emission would be high. On the other hand, the adoption of EGR (used with stoichiometric mixtures) to dilute the engine intake mixture lowers the NOx levels, but also deteriorates combustion quality.

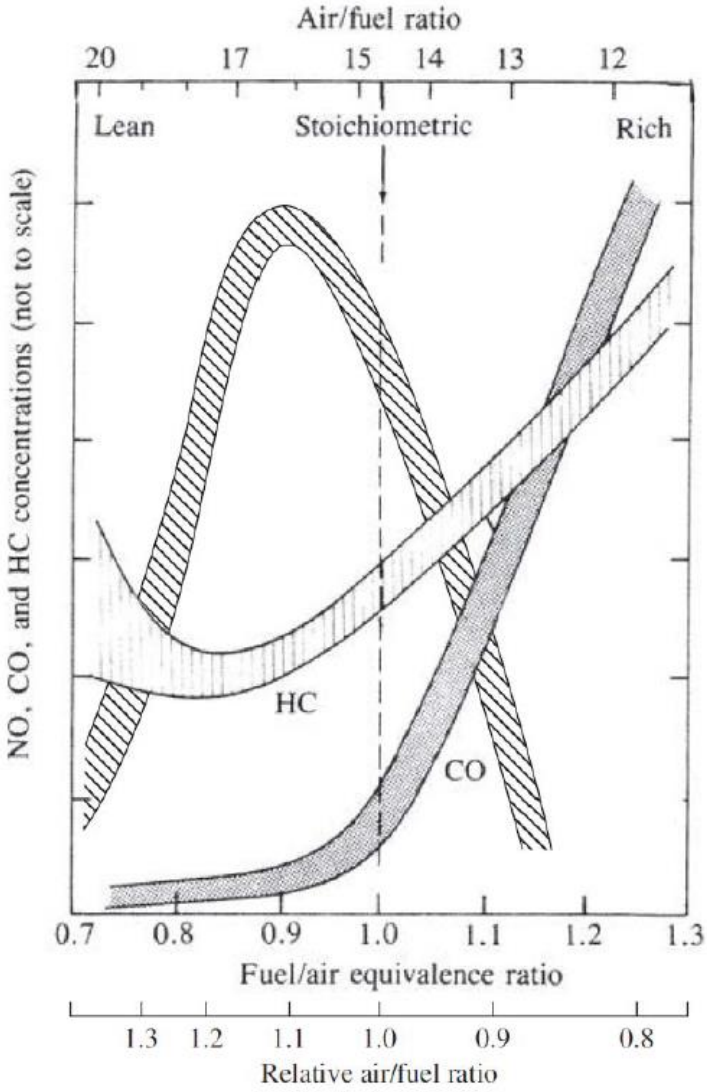

Figure 4 - Variation of HC, CO and NO concentration in the exhaust of a conventional SI engine with relative air/fuel ratio and fuel/air equivalence ratio [24]

As soon as power and emissions control goals are conflicting, control techniques are required to adapt the air/fuel ratio over all engine operating modes, in order to accommodate power requirements when needed and to reduce emissions whenever high power is not requested, by always keeping into account that the three-way catalytic converter show a good conversion efficiency for all the three main pollutants only in a narrow A/F ratio window, in correspondence of stoichiometric A/F ratio.

Figure 5 - Gasoline Engine ECU Control ModesFigure 5 - Gasoline Engine ECU Control Modesshows the control algorithm implemented by the ECU, taking into account the operating conditions of the engine and special requirements.

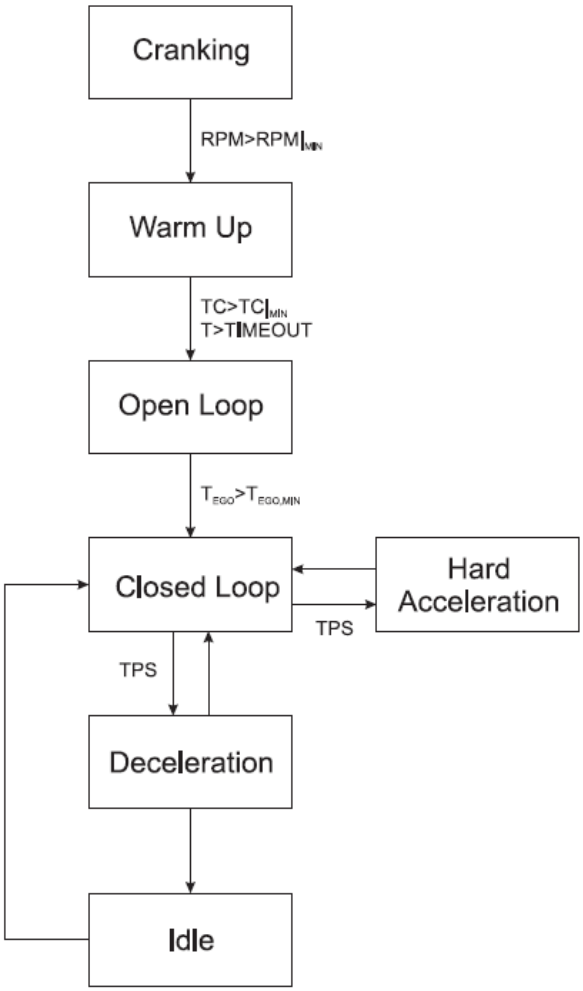

Figure 5 - Gasoline Engine ECU Control Modes [25]

The first phase depicted, i.e., cranking phase, is the operating mode when the vehicle is turned on and the engine is required to start. In this first condition, ECU provides to communicate to fuel injectors actuators to deliver a rich mixture to the cylinders.
As soon as engine speed exceeds a specific threshold, the ECU registers this information and switches the engine state from cranking to warm up, a period when the A/F ratio is always kept under stochiometric. This state is left whenever the coolant system temperature exceeds the nominal temperature: at this point, the open-loop state is reached and the engine results to operate under nominal conditions.

Open-loop A/F control coincides with a phase when data related to the oxygen concentration in the exhaust (gathered by λ sensor) are still not available, since the temperature of the sensor have not reached its nominal working temperature yet. The assessment of the A/F ratio, necessary to

control the nature of the injected mixture, takes place in this phase by an algorithmical evaluation on the basis of the indirect information provided by other sensors (i.e., MAF and rpm sensor). Of course, the accuracy of A/F evaluation is strongly affected by the nature of this methodology.

As soon as the λ sensor actually reaches the nominal temperature, data related to oxygen level in exhaust become reliable and a correct approximation to stochiometric ratio is provided. This phase, indicated as closed loop A/F control, results, as previously explained, into a reduction in terms of pollutant emissions and into a proper catalytic converter efficiency. On the basis of the TPS, closed loop mode can be left to achieve hard acceleration or deceleration mode (and eventually idle, if rpm is reduced under a specific threshold). In the first case, a rich mixture is injected (power delivery is here the main target), while in the second one, A/F ratio is increased, in order to achieve lower fuel consumption.

A wider explanation regarding Open and Closed Loop modes will be provided in following sub-chapters.

3.2.1 Open-Loop mode

The Open-Loop control is a configuration which does not monitor or measure the condition of its output signal and there is no feedback; this is a type of control system in which the output has no influence or effect on the control action of the input signal, therefore, an open-loop system is expected to faithfully follow its input command or set point regardless the final result, as depicted in Figure 6:

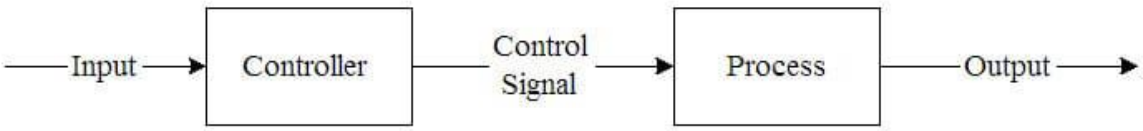

Figure 6 - Open-loop control scheme [26]

In case of engine control system, this is the control type which is implemented by the ECU until the operating temperature of the EGO sensor does not overcome a specific threshold. Until that moment, the optimal injection timing, and consequently the optimal fuel amount to be injected in the cylinders, is evaluated linearly starting from two inputs, related to the operating point analysed: rpm of the engine and air-mass-flow in the intake manifold.

By considering that in Open-Loop phase the A/F ratio of the mixture should be equal to stoichiometric one:

$$\frac{A}{F} = \frac{A}{F}_{stoichiometric} \tag{1}$$

It is possible to say that the mass of fuel to be injected must correspond to:

$$F^{opt} = \frac{A}{\frac{A}{F}_{stoichiometric}} \tag{2}$$

If the ECU receives the signal from the MAF sensor, it knows the mass air flow $\dot{m}_a$ from the intake manifold, from which follows that, among the engine period T (= $\frac{60}{n}$, with n rpm of the pistons), the mass of air A entering the intake manifold equals $\dot{m}_a T$. In a four cylinders engine, two cylinders undergo the intake phase within a period, which means:

$$A = \frac{\dot{m}_a T}{2} = \frac{60\dot{m}_a}{2n} \tag{3}$$

$$F^{opt} = \frac{60\dot{m}_a}{2n * \frac{A}{F}_{stoichiometric}} \tag{4}$$

If R_f is the fuel injection rate, it is possible to say that the injection time to achieve a stoichiometric mixture can be assessed as:

$$T_{inj,OL} = \frac{F^{opt}}{R_f} = \frac{60\dot{m}_a}{2n * R_f * \frac{A}{F}_{stoichiometric}} \tag{5}$$

Based on the results of such calculations, the injector opening signal, which drives the injector drivers, is generated by the ECU.

As soon as the output directly depends on MAF and rpm estimations, is it clear that inaccuracies are unavoidable. These will be overcome by means of Closed-Loop control mode.

3.2.2 Closed-Loop mode

The Closed-Loop control is a configuration where a portion of the output signal is fed back to the input to reduce errors and improve stability. Closed-Loop systems are designed to automatically achieve and maintain the desired output condition by comparing it with the actual condition. It does this by generating an error signal which is the difference between the output and the reference input, then this signal is fed to the controller so as to reduce system errors and bring back the output of the system back to the desired output.

Moreover, because a closed-loop system has some knowledge of the output condition by means of the sensor, it is better equipped to handle any system disturbances or changes in the conditions which may reduce its ability to complete the desired task.

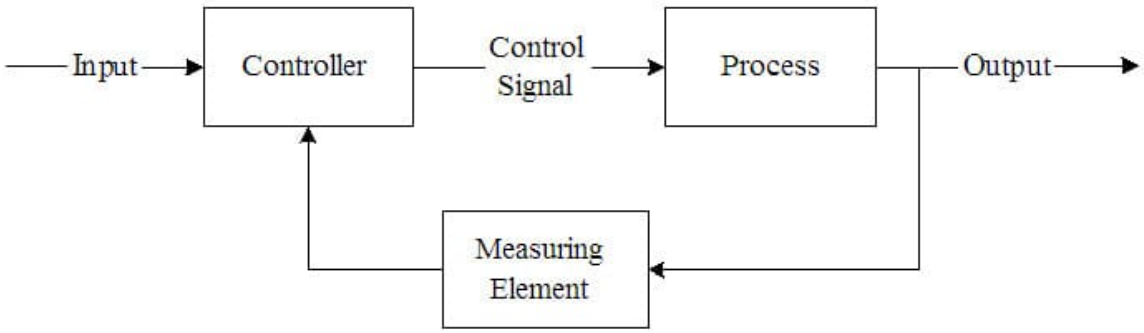

Figure 7 - Closed-loop control scheme [27]

In Closed-Loop engine control mode, whose scheme is depicted in Figure 7, the sensor which permits to assess the error function is the oxygen sensor, which functioning principles will be analysed in Chapter 3.3.3.

The three main types of feedback control are:

1. Proportional Control;

2. Integral Control;
3. Proportional-Integral Control;

Listed on the basis of the increasing effectiveness in making the output close to the desired value. As soon as in this project an Integral Control method has been used in order to achieve a Closed-Loop control, this only will be described.

As previously discussed, the Closed-Loop mode is enabled as soon as the EGO sensor information becomes reliable, since its nominal operating temperature is achieved. In this mode, the ECU, after gathering information from sensor, assesses if the mixture is rich (in that case the injector opening time must be reduced, in order to achieve a stoichiometric ratio), or lean (in that case the injector opening time must be increased).

The Closed-Loop injection timing applied at the n-th cycle can be assessed as:

$$T_{inj,CL} = T_{inj,OL}(n)[1 + C_L(n)] \tag{6}$$

where $C_L(n)$ is a correction factor which takes into account the information provided by the EGO sensor, which can be expressed in terms of the variable $EGO(n)$:

$$EGO(n) = \begin{cases} -1 \; for \; \lambda > 1 \; (lean \; mixture) \\ +1 \; for \; \lambda < 1 \; (rich \; mixture) \end{cases} \tag{7}$$

$$C_L(n) = k_I \sum_{i=0}^{n-1} EGO(i) \tag{8}$$

In order to achieve negative feedback, the constant k_I must be less than zero.

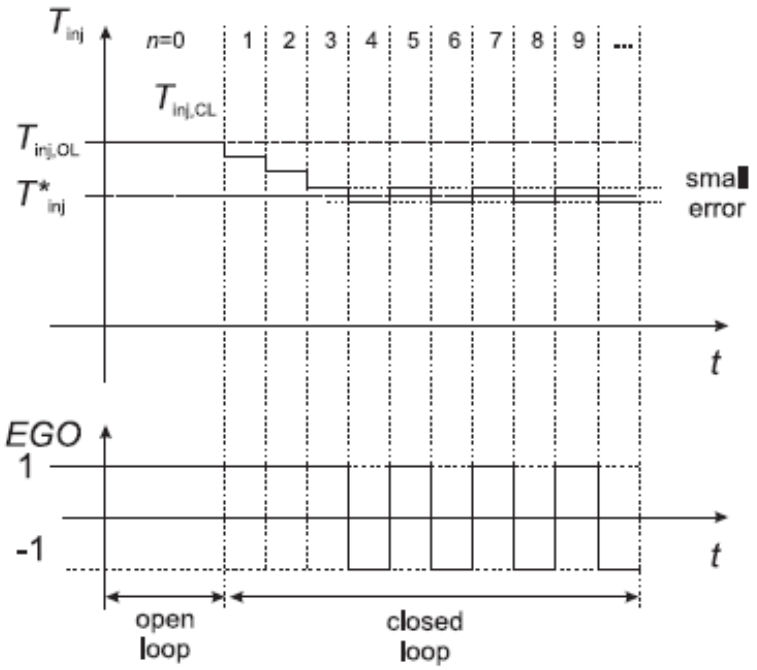

Figure 8 - Integral closed-loop A/F ratio control [28]

Figure 8 shows how Closed-Loop control is intended to work. If cycle $n = 1$ is considered, the correction of the injection time due to the correction term $C_L(1)$ (based on the information retrieved from EGO sensor ant cycle $n = 0$) equals $k_I T_{inj,OL}$. Since such a correction is not sufficient to switch to lean mixture (EGO is still +1), in the following cycle $C_L(2)$ will be equal to $2k_I T_{inj,OL}$, twice with respect to previous cycle. The correction term is increased in absolute value at each cycle, the injection time is then reduced until the lean mixture is reached: at that moment, a negative EGO is detected and added to $C_L(n)$ and the injection time is increased.

3.3. Sensors working principles

3.3.1 RPM sensors

The RPM sensor is up to detect the crankshaft rotational speed: its data are gathered and processed by ECU in order to switch from cranking state to warm-up and then from deceleration mode to idle.

Typical devices used to sense shaft speed are optical sensors and magnetic sensors. These devices work by sensing speed data in the form of pulses.

3.3.1.1 Optical sensors

These elements, also known as encoders, output a pulse string in response to the amount of rotational displacement of a shaft. A separate counter counts the number of output pulses to determine the amount of rotation based on the count; the counter is reset at the reference position and the number of pulses from that position is added cumulatively. When a disk with an optical pattern revolves along with the shaft, light passing through two slits is transmitted or blocked accordingly. The light is converted to electrical currents in the detector elements, which corresponds to each slit, and is output as two square waves.

Figure 9 provides a simplified scheme of the system.

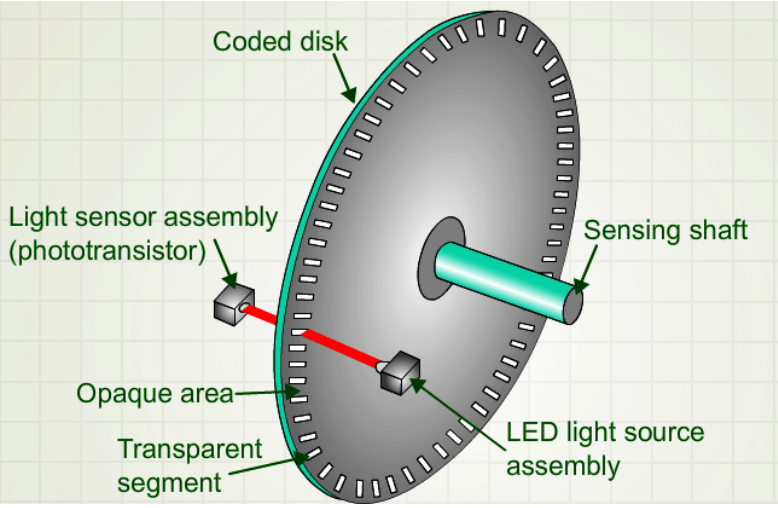

Figure 9 - Encoder scheme [29]

Since slits are equally spaced on the disk, it is possible to say that the angular displacement of the rotor plate, integral with the crankshaft, is inversely proportional to the number of slits; by considering the number of times the light actually passes through the disk and reaches the phototransistor within a specified period of time, it is possible to evaluate crankshaft angular speed.

3.3.1.2 Magnetic sensors

These devices use the variable reluctance magnetic sensing principle, whereby a cylindrical permanent magnetic core, with a coil wire wound around it, mounted on a stationary hub carrier, produces a magnetic field which overlaps the rotating excitor ring. The excitor may be of the tooth ring or rib-slot ring type attached to the rotating wheel hub or drive shaft. These teeth or slots are arranged radially and determine the frequency of the signal transmitted to the ECU, with the speed of rotation of the road wheel. This phenomenon results from teeth or gaps of the excitor passing through the magnetic field of the sensor as the wheel and excitor revolve; the changing intensity of the magnetic field is detected by the coil wrapped around the magnetic cone and so an alternating voltage is induced into it, whose frequency is proportional to the speed of the rotating wheel. Figure 10 Figure 10 - Magnetic speed sensor and excitorshows a scheme of a magnetic speed sensor with its excitor.

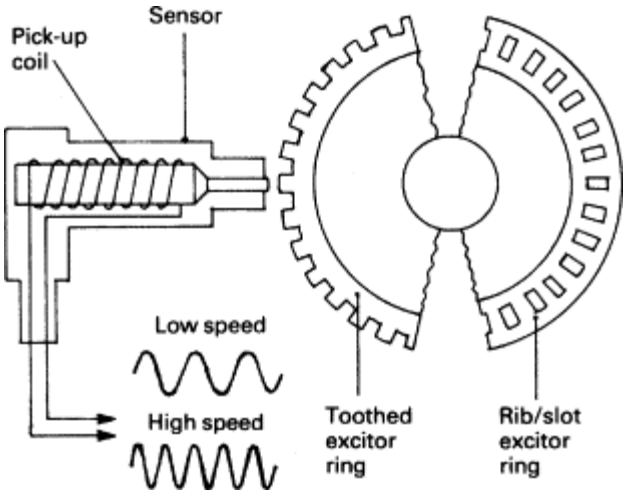

Figure 10 - Magnetic speed sensor and excitor [30]

3.3.2 Temperature sensors

Most common temperature sensors employed in automotive field are:

- Thermistors;
- Thermocouples;
- Semiconductor Temperature Sensors.

First typology will be analysed only, since thermocouples and semiconductor temperature sensors are not used for control purposes.

Thermistors are resistive elements whose resistance is affected by changes in temperature. It is possible to distinguish between two main typologies of thermistors: PTC (Positive Temperature Coefficient, i.e, resistance increase with temperature) and NTC (Negative Temperature Coefficient, i.e. resistance decrease with temperature). In both cases, the thermistor is connected in series with a temperature-independent resistance (R_0, which causes the circuit to behave as a temperature-dependant voltage divider), as depicted in Figure 11.

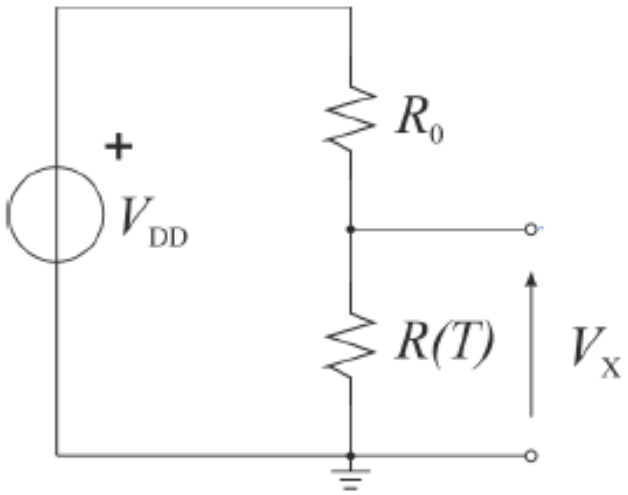

Figure 11 - Thermistor circuit [31]

By processing V_x and converting it into digital, by means of a look-up table in the ECU, it is possible to find the corresponding temperature.

3.3.3 EGO sensor

As previously described, being able for the ECU to switch from one engine state to another is what permits to effectively control fuel consumption, pollutants emission and power demand. To perform this activity, a continuous stream of information coming from the exhaust is necessary to understand the nature of the mixture injected in combustion chambers.

It is possible to define λ as:

$$\lambda = \frac{\alpha}{\alpha_{st}} \approx \frac{\alpha}{14.6} \tag{9}$$

Where α is the air/fuel ratio.
A EGO sensor (also known as EGO sensor) is an electronic component typically mounted to the exhaust system tube or on the side of the catalytic converter, with the sensor part inside the tube;

this measures the difference between oxygen concentration which is present in the atmosphere and that present in exhaust gases and send this information to the ECU.

It is possible to distinguish between two sensor typologies:

- HEGO sensors (i.e. Heated Exhaust Gas Oxygen sensors): first kind to be used, they give no information regarding the actual value of λ, the output is a boolean type, to indicate whether the mixture is lean or rich (1 or 0);
- UEGO sensors (i.e. Universal Exhaust Gas Oxygen sensors): they provide a variable current value depending on λ value. The system's goal is that of targeting the precise value of λ, but works efficiently whenever this value does not change rapidly.

The most used typology of EGO sensor is that made of Zirconium Dioxide. The external surface of the element made of ZrO_2 is directly in contact with exhaust gases, while the internal surface is in contact with the atmosphere. Both surfaces are covered by a thin platinum layer. Oxygen ions go through the ceramic layer and charges electrically the platinum, which behaves as an electrode: the electrical signal generated is retrieved by the connection wire exiting the sensor. The ZrO_2 element becomes permeable to oxygen ions more or less at a temperature equal to 300°; when the concentration of oxygen is different on the two sensor surfaces, a voltage is generated, which results into a low signal when the mixture is lean, and into a high signal when it is reach. Typically, the change in the signal intensity takes place when A/F ratio is 14.7 and it is called "EGO1". This ratio is also considered to correspond to a complete combustion.

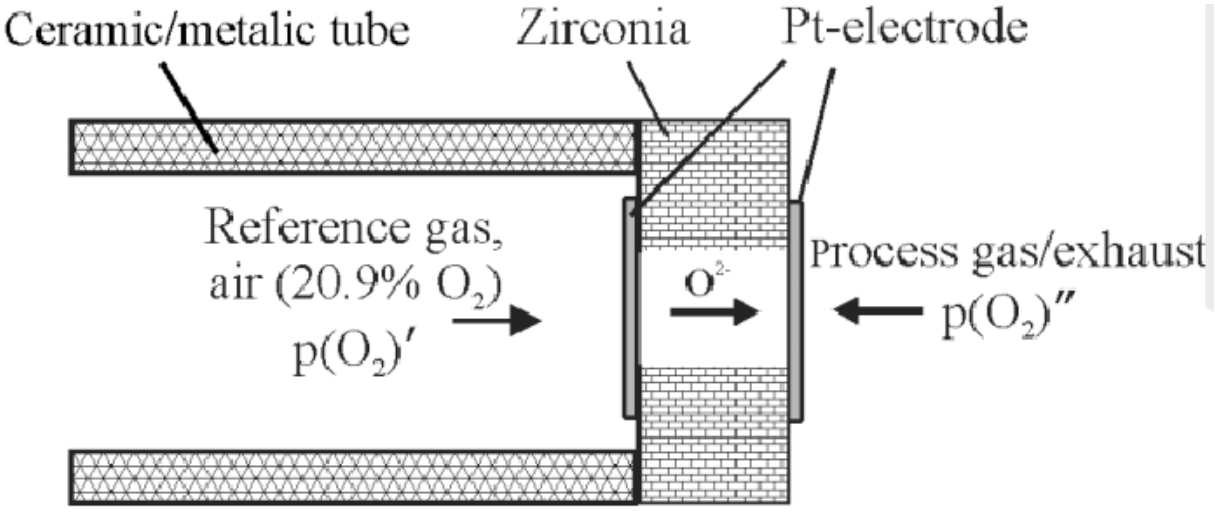

Figure 12 - Zirconium dioxide EGO sensor [32]

Figure 12 Figure 12 - Zirconium dioxide EGO sensorshows a scheme of a EGO sensor with zirconium dioxide plates.

3.3.4 Throttle Position Sensors

A potentiometer is a passive electronic component, which can be defined as a three-terminal resistor having either sliding or rotating contact, which forms an adjustable voltage divider.

The potentiometer consists of a long resistive wire and a battery, whose EMF voltage is known, used as a driver cell voltage (input voltage depicted in Figure 13). A primary circuit arrangement can be obtained by connecting the two ends of the resistive wire to the battery terminals; then, one end of the primary circuit is connected to a cell whose EMS has to be measured (output voltage, as depicted in Figure 13), forming a secondary circuit.

Error! Reference source not found.

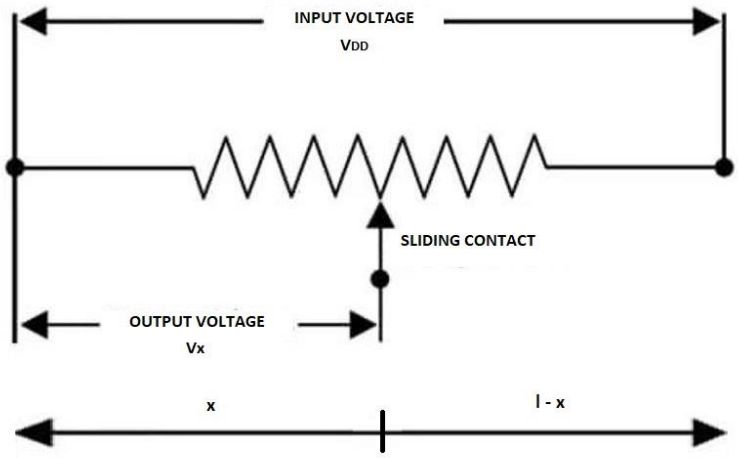

Figure 13 - Potentiometer functioning scheme

The functioning principle of this circuit is that the voltage drop across any part of the uniform resistive wire is directly proportional to the length of the wire if a constant electric current is flowing through it. The position of the sliding contact is varied across the length of the wire,

dividing it into two different resistances, both defined by means of the second Ohm's law (*10*) and (*11*):

$$R_1 = \frac{\rho(l-x)}{S} \tag{10}$$

$$R_2 = \frac{\rho x}{S} \tag{11}$$

By means of voltage divider formula it is possible to define V_x as described in (*12*) and (*13*):

$$V_x = \frac{R_2}{R_1 + R_2} V_{DD} \tag{12}$$

$$V_x = \frac{x}{l - x + x} V_{DD} = \frac{x}{l} V_{DD} \tag{13}$$

As soon as V_{DD} is known, as well as l, it is possible to identify a direct proportionality between the position of the sliding contact and the output voltage V_x.

In regard to angular position sensor typology, same considerations can be done, but in this case the position of the sliding contact x must be considered to be an arc of angle α, while the length of the resistive wire could be indicated as a semi-circumference, as described in (*14*) and (15):

$$x = \alpha r \tag{14}$$

$$l = \pi r \tag{15}$$

From (*14*) and (15) follows that:

$$V_x = \frac{\alpha r}{\pi r} V_{DD} = \frac{\alpha}{\pi} V_{DD} \tag{16}$$

Also in this case, it is possible to identify a direct proportionality between the calculated voltage and the angular position of the sliding contact.

A potentiometer assembled on the Arduino board results to accurately replicate the actual functioning of the TPS, excluding of course the technical specification of Arduino component, which will be discussed in Chapter 4.

3.3.5 Mass Air Flow sensors

The Mass Air Flow sensor (i.e., MAF sensor) generally used for automotive purposes is a hot wire sensor, whose functioning is based on heat dissipation by convection in a resistive element and permits to determine the mass of air flowing into engine's air intake system. The working principle of this sensor exploits the Joule effect: when a constant voltage V is applied to a resistor, this dissipates an electric power ($P_{diss} = \frac{V^2}{R}$) which is converted into heat and dissipated into the external environment (initially at T_{amb}), increasing its temperature by $\Delta T = P_{diss} R_{TH}$, where R_{TH} is the thermal resistance between the resistor and the air. If we consider that the heat is exchanged by convection, it results that the amount of heat ΔQ exchanged between the resistor and the environment can be expressed as the difference between the heat in the mass of air transferred to the environment Q_{out} and the heat in the mass of air coming from the environment Q_{in}, from which follows that:

$$\begin{aligned} \Delta Q = Q_{out} - Q_{in} &= c_p m_a T - c_p m_a T_{amb} \\ &= c_p m_a \Delta T \end{aligned} \tag{17}$$

And, by taking the time derivative of the previous equation:

$$P_{diss} = \frac{d\Delta Q}{dt} = c_p \dot{m}_a \Delta T \tag{18}$$

Which shows that $\dot{m}_a$ (mass air flow) from the resistor to the environment is related with the dissipated thermal power.
From previous equation, it follows that:

$$P_{diss} = \frac{V^2}{R} = c_p \dot{m}_a \Delta T \tag{19}$$

$$V = \sqrt{R c_p \dot{m}_a \Delta T} \tag{20}$$

To exploit such a principle, ΔT must be known and/or constant. To this purpose, the circuit depicted in Figure 14 is employed.

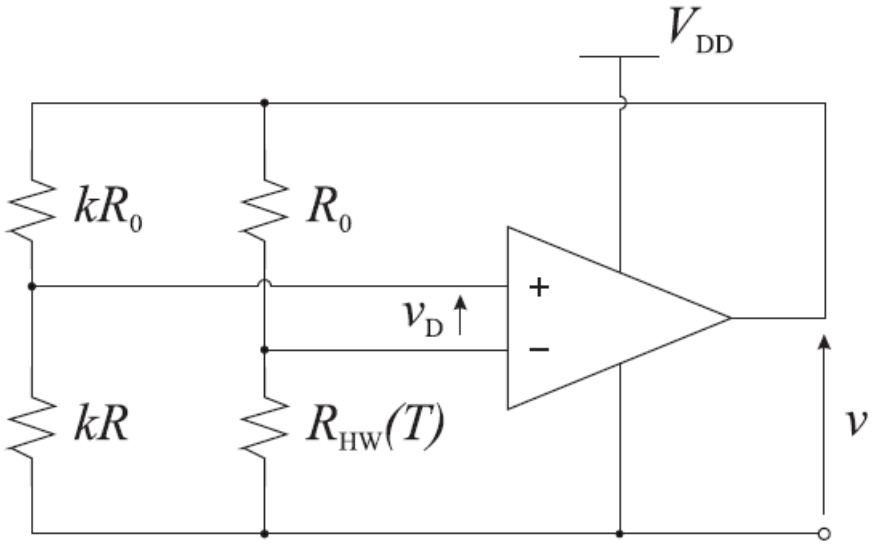

Figure 14 - MAF sensor circuit [33]

Here, the hot wire is made of a material whose resistivity increases with temperature and therefore can be described in terms of thermal resistance $R_{TH}(T)$ which increases with temperature.
A wide description of the above circuit working principle is out of the scope of this discussion; it is enough to consider that the opamp (operational amplifier) output voltage v depicted in Figure 14 is proportional to mass air flow and to quantities accurately compensated so that to result constant:

$$v = (1 + \frac{R_0}{R(T_{amb})}) \sqrt{R(T_{amb}) c_p \dot{m}_a (T - T_{amb})} \quad (21)$$

Which permits to properly correlate the mass of air flowing through the cylinder and the voltage signal which will be then sent to the ECU.

4. Arduino microprocessor characteristics and main functions

Arduino first came around in early 2000s, when students at Interaction Design Institute of Ivrea, Italy, were trying to develop a low-cost microcomputer which would allow people to pursue many technical projects that would normally be prohibitively difficult or expensive for people who are less familiar with advanced electronics.
This was coupled with the development of easy-to-understand platforms which would allow people to have the opportunity to learn more about the complexity of hardware and software in an intuitive manner that would have a relatively low learning curve and enable them to get into tinkering and coding.
What resulted was a tool that serves as an amazing prototyping device. The Arduino is capable of allowing people to easily build prototypes of electronic systems at little costs in terms of components. It is also relatively user error-tolerant of a platform and allows users the room to experiment in a low-stakes environment.
Moreover, the working language of Arduino is relatively simple: many of the electronics which are compatible with this platform are very easy to use and there is a large enough community which can provide support to beginners.
Arduino Uno board (the one adopted for this project) consists of an ATmega328P microcontroller, which is a high performance, low power control from Microchip.

This chapter is intended to provide a general functional overview of Arduino Uno board, with a major emphasis on connector pinouts.

Figure 15 shows the top view of an Arduino Uno board.

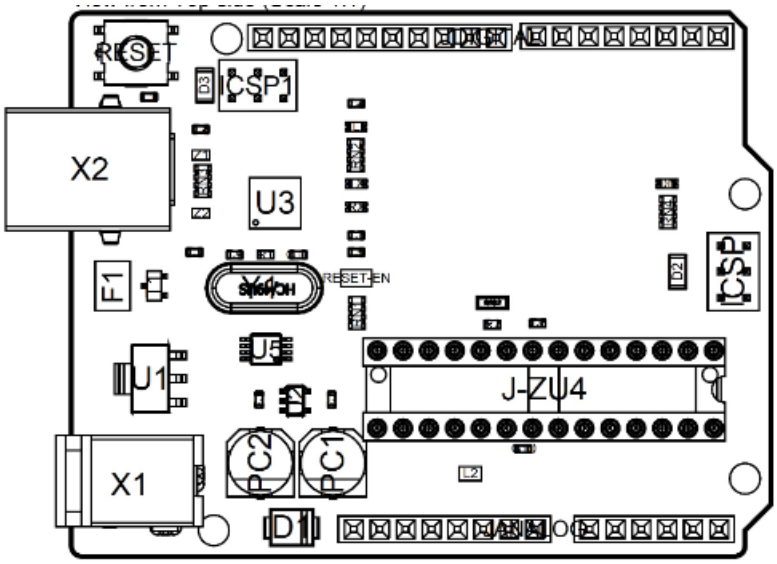

Figure 15 - Arduino Uno top view [34]

Main technical specifications of ATmega328P are depicted in Table 2 [35]:

Core Processor	AVR
Core Size	8-Bit
Speed	20MHz
Connectivity	I^2C, SPI, UART/USART
Number of I/O	23
Program Memory Size	32 KB (16K x 16)
Program Memory Type	FLASH
RAM Size	2K x 8
Voltage – Supply (Vcc/Vdd)	1.8 V ~ 5.5 V
Data Converters	A/D 8x10b
Oscillator Type	Internal

Operating Temperature	-40 °C ~ 85 °C

Table 2 - ATmega328P technical specifications

A complete and extensive discussion about hardware characteristics is out of the scope of this essay; the main emphasis will be put on pins and components which have been actually used during the design phase of the prototype: further descriptions and explanation will be provided about these.

Figure 16 - Arduino Uno analog, digital and PMW pins [36]

Figure 16 displays Arduino Uno pins: digital pins are the ones highlighted by the red rectangle in, numbered from 0 to 13; analog pins are that indicated by yellow rectangle, while PMW (Pulse-Width Modulation) pins are that highlighted by green rectangles.

- DIGITAL PINS:

  Almost every Arduino board is embedded with 14 digital pins: these can be used both in INPUT mode (to acquire digital signals) and in OUTPUT mode (to send logical signals to another board or, like in this case, via serial communication to MATLAB). As soon as these impulses are of digital types, the value these pins can receive/send corresponds to 0/1, or, in Arduino language, to LOW/HIGH voltage.

`pinMode(pin,Mode)` function allows to set a certain pin to the correct mode, INPUT or OUTPUT, and stands for all types of pins. To send a digital value, the function `digitalWrite(pin,level)`, where the level can be LOW or HIGH, must be used; on the other hand, to read a digital value, the pinMode must be set to INPUT and the digital command `digitalRead(pin)` must be implemented.
In this project, `digitalRead(pin)` function has been used, to read the slide switch signal (i.e., engine ON/OFF state).

- ANALOG PINS: Arduino Uno board contains a A 6-channels, 10kS/s, 10-bit analog-to-digital converter. This means that it can map input voltages by using integers in a range going from 0 to 1023; the resolution of the lectures can be considered than to equal 5/1024 per unity or 4.9 mV per unity. To read an analog input, more or less 100 microseconds are necessary, which involves that the maximum lecture rate is more or less 10.000 times per second. The microcontroller reads voltage signal and converts it into a number between 0 and 1023, by means of the analog-to-digital converter. By taking the potentiometer as an example, when the shaft is turned all the way in one direction, the pin voltage is 0, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, `analogRead()` returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

- PWM PINS: a PWM signal is a square-wave characterized by the frequency (fixed) and by the duty cycle (variable), which is the ratio between the time the square wave assumes a high value and the period. Basically, this kind of modulation is used in communication protocols where information is codified in terms of time duration of each impulse. In many micro-controllers, PWM signals can be low-pass filtered to extract their DC component: this is implemented by means of timers which are able to generate rectangular waves and that are properly programmed so that to change their duty cycle and consequently the voltage value obtained as an output.

4.1. Potentiometer

As previously said, the role of the potentiometer is that of simulating TPS working principles. Students will have the possibility to change its angular position and to appreciate both the MATLAB plot related to sensed voltage and eventually the engine state switching.

The adopted version of potentiometer is CA6: this is a 6mm carbon component with plastic housing and dust-proof protection. Terminals are manufactured in tinned brass to guarantee better soldering and higher resistance corrosion.

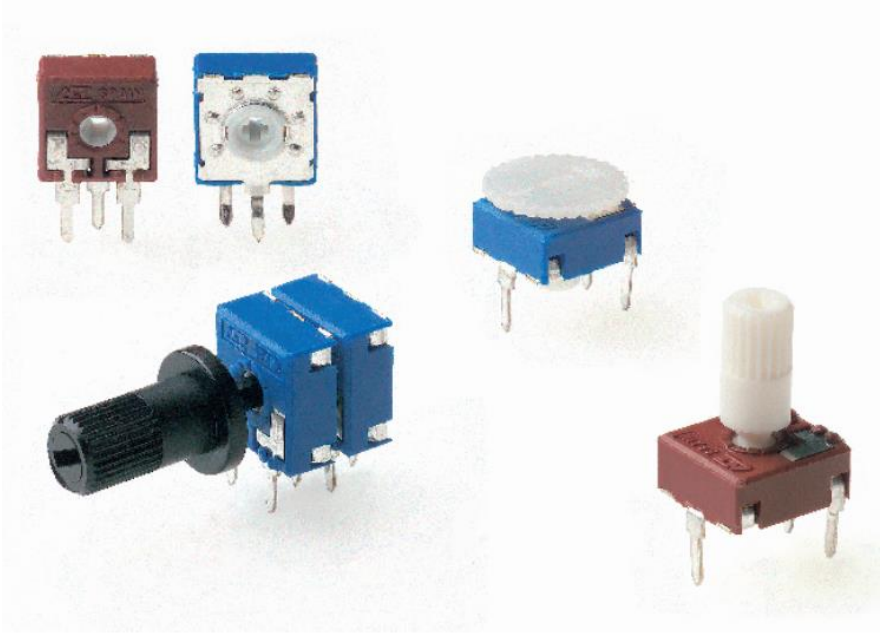

Figure 17 – Potentiometers [37]

By default, the rotor has the geometrical characteristics shown in Figure 18:

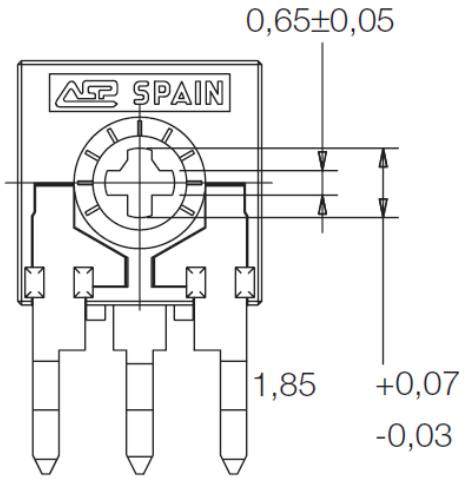

Figure 18 - Potentiometer top view geometry [38]

While shafts can differ for shape, height and thickness.

Terminals, are always recommended with “snap in”, in order to better hold the component to the board prior to soldering, as depicted in Figure 19:

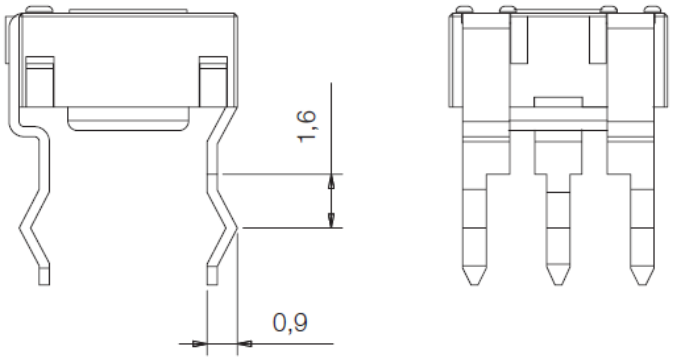

Figure 19 - Potentiometer side view geometry [39]

The mechanical angle or rotation of CA6 potentiometer is 235° +/- 10°, while the electrical one is 215° +/- 20°.

4.2. Slide switch

Slide switch actuator has been used to simulate ON/OFF states of the engine: the rated voltage of this component stands in a range which goes from 12 V up to 24, and a travel to switch from ON state to OFF state equal to 1.6 mm.
Figure 20 depicts geometrical characteristic of the used slide switch.

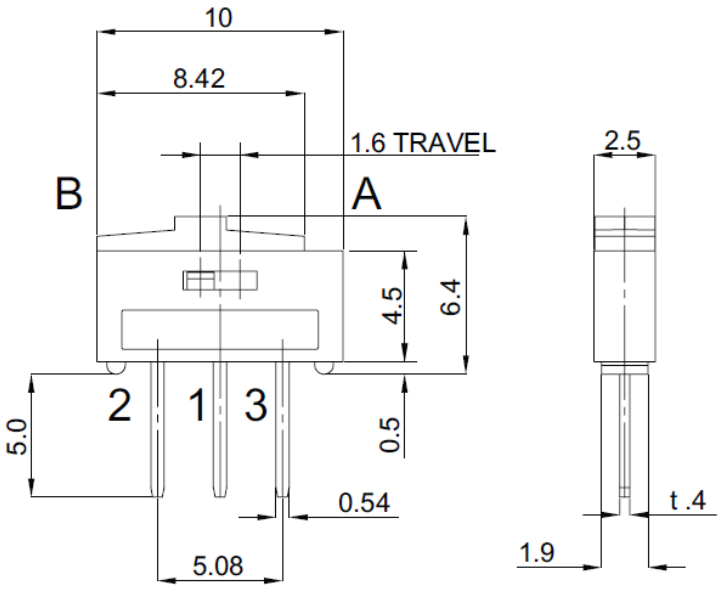

Figure 20 - Slide switch geometry [40]

There are mainly four types of switches in Arduino:

- SPST Switch (Single Pole Single Throw), with one input and one output; in this case, the circuit is ON when the switch is closed and vice versa.

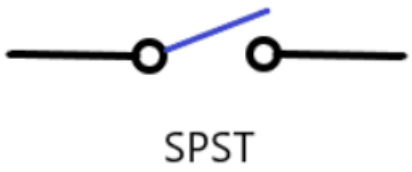

Figure 21 - SPST switch scheme [41]

- SPDT Switch (Single Pole Double Throw), with a single input which can switch between two outputs.

Figure 22 - SPDT switch scheme [42]

- SP3T Switch (Single Pole Three Throw), with one input and three outputs, where each input corresponds to any of the output in a circuit.

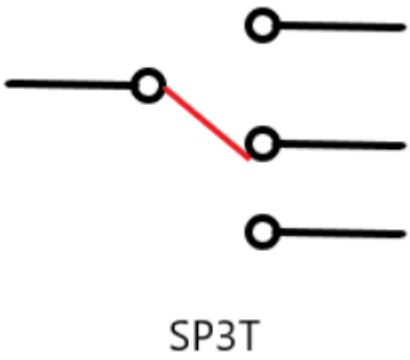

Figure 23 - SP3T switch scheme [43]

- DPDT Switch (Double Pole Double Throw), whit two inputs and four outputs; each input of a switch in Arduino can be connected to either of the two outputs.

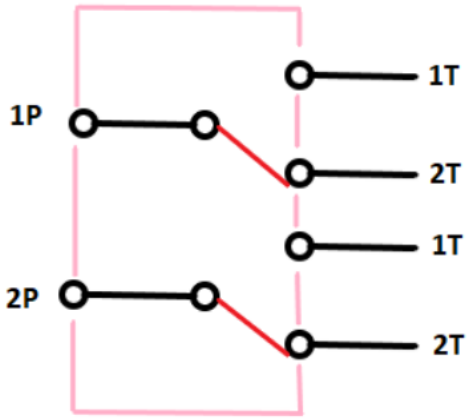

Figure 24 - DPDT switch scheme [44]

The type adopted in this project is the simplest one, the SPST.

5. Prototype development

This chapter is intended to explain the core logic of this project: a general overview over design purposes and choices made will be provided, together with explanations regarding serial communication principles which have been here implemented, sensors simulation and injection times assessments.

The purpose of the project, as previously mentioned, is that of providing students an educational tool which may give them the possibility to physically experience ECUs basic working principles: specifically, that related to engine control.

In order to design the prototype and to replicate an almost realistic behaviour of an ECU, several sensors have been simulated: four sensors (i.e., rpm sensor, coolant temperature sensor, EGO temperature sensor and MAF sensor) have been virtualized in MATLAB, while other two sensors (i.e., ON/OFF engine sensor and TPS) are implemented in hardware by a slide switch and a potentiometer assembled in Arduino.
In first place, Arduino detects the voltage of the slide switch: HIGH value stands for engine ON, while LOW value stands for engine OFF. As soon as the system is sensed to be on ON state, Arduino starts detecting the potentiometer voltage and stores the values; engine ON information is then sent, by means of serial communication (which will be discussed in more detail in Chapter 5.1), to MATLAB, which starts the other sensors simulation, by means of the implementation of transient functions.
Data from MATLAB are then sent back to Arduino, which implements the state machine, which means that it compares sensors values (including potentiometer ones) with specific thresholds and switches from one state mode (i.e., cranking, warm-up, open-loop, closed-loop, hard acceleration, deceleration and idle) and stores this information into a variable called “state”, again sent back to MATLAB so that the software may display it.
On the other hand, two sensors in particular are necessary for the evaluation of the open loop injection time, according to Equation (5), which are rpm sensor and MAF sensor. The open-loop injection time is sent on the serial and received by MATLAB, which assesses the optimal injection time as the sum of the received open-loop one plus an error function and then compares it with the closed-loop injection time of the previous cycle and simulates the EGO sensor, by returning an EGO value equal to -1 whenever the closed-loop time is sensed to be less than the optimal one

and an EGO value equal to +1 whenever the closed-loop time is sensed to be larger than the optimal According to Eq. (8) and Eq. (6), the closed-loop time is calculated as a correction of the open-loop time, on the basis of the EGO value detected in the previous cycle, so that the fuel injection time can be closer to the optimal value.

On the other contrary, whenever the slide switch voltage is LOW and the engine is on OFF state, the signal is sent to MATLAB as well, which implements again sensors simulation, by taking into consideration the proper expected behaviour (speed goes to zero, temperature sensors start to decrease to reach ambient temperature, etc.)

Figure 25 and Figure 26 summarize the logical structure implemented by the prototype to simulate the state machine and to assess the fuel injection times, respectively.

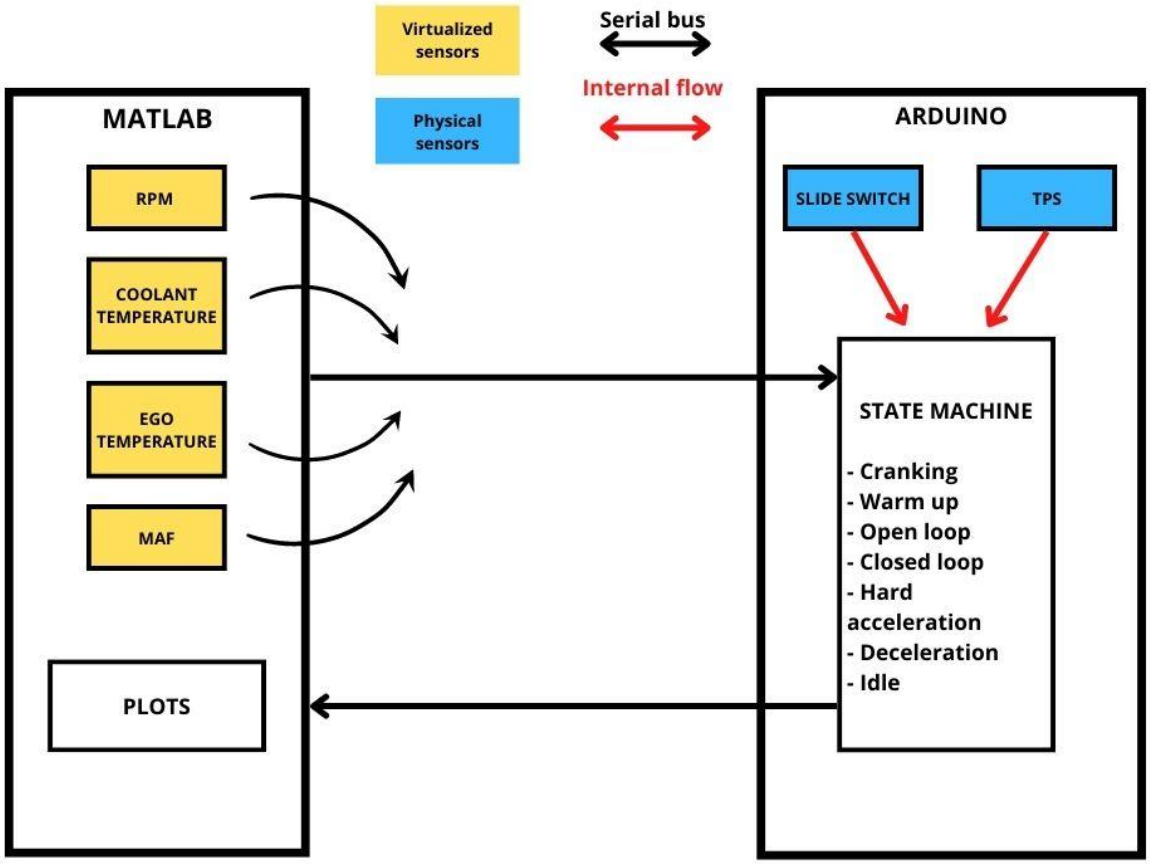

Figure 25 – State machine logical structure implemented

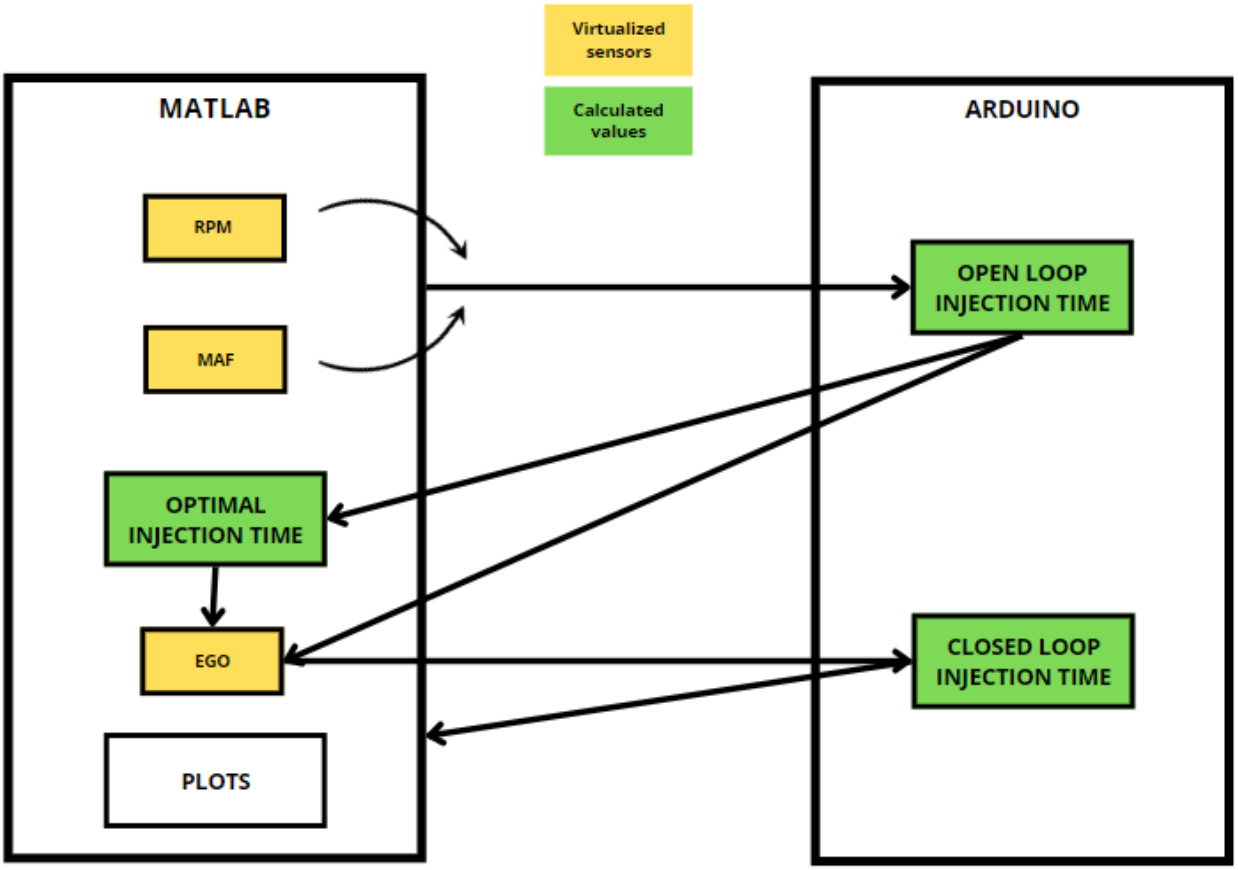

Figure 26 - Injection times assessment logic flow

A real-time counter is configured in Arduino so that to run the whole procedure (sensor acquisition, FSM update and output calculation) once per second. This has been implemented by means of a specific library ("elapsedMillis") which permits to assess the time needed from the Arduino software to implement the loop.

Table 3 - Time sequence of actions implemented by the prototypesummarizes the time sequence of the actions implemented:

Action number	Software implementing action	Description
1	Arduino	Slide switch state reading
2	Arduino	Potentiometer voltage reading

3	Arduino	Printing on the serial the of engine state (ON/OFF)
4	MATLAB	Reading of the data sent by Arduino
5	MATLAB	Sensors virtualization (rpm, coolant temperature, EGO temperature, MAF)
6	MATLAB	Printing of sensors data on the serial
7	Arduino	Sensors data acquisition and implementation of switch state function (state machine)
8	Arduino	Open loop timing calculation, on the basis of rpm sensor and MAF sensor
9	MATLAB	Reading of engine mode and of open-loop injection time
10	MATLAB	Calculation of optimal injection time, as the sum of open-loop injection time and the error function
11	MATLAB	Printing of optimal injection time on the serial
12	Arduino	Optimal injection time acquisition and virtualization of EGO sensor. Calculation of

		closed-loop injection time and printing of this on the serial
13	MATLAB	Data acquisition, displaying of engine mode and plotting of sensors' behaviour and injection times.

Table 3 - Time sequence of actions implemented by the prototype

From the hardware point of view, the connections to guarantee the communication between interfaces are of different types.
In first place, as shortly described in Chapter 4, slide switch and potentiometer are assembled on the Arduino board by means of their three terminals: for both of them, lateral terminals are connected to a voltage source (5V) and to ground reference voltage; the central terminal, on the other hand, is the one devoted to the transmission of the sensor's signal and is connected, in case of the slide switch, to a digital pin, while in case of the potentiometer, to an analog pin. This is due to the fact that, as already mentioned, slide switch logical output is the ON/OFF engine state and digital pin can return only a value which corresponds to 0 or 1, while in case of the potentiometer it is needed that the voltage is detected within a specific range (0 to 5V) and in this case an analog pin is needed.

Figure 27 shows the realized circuit on the Arduino board.

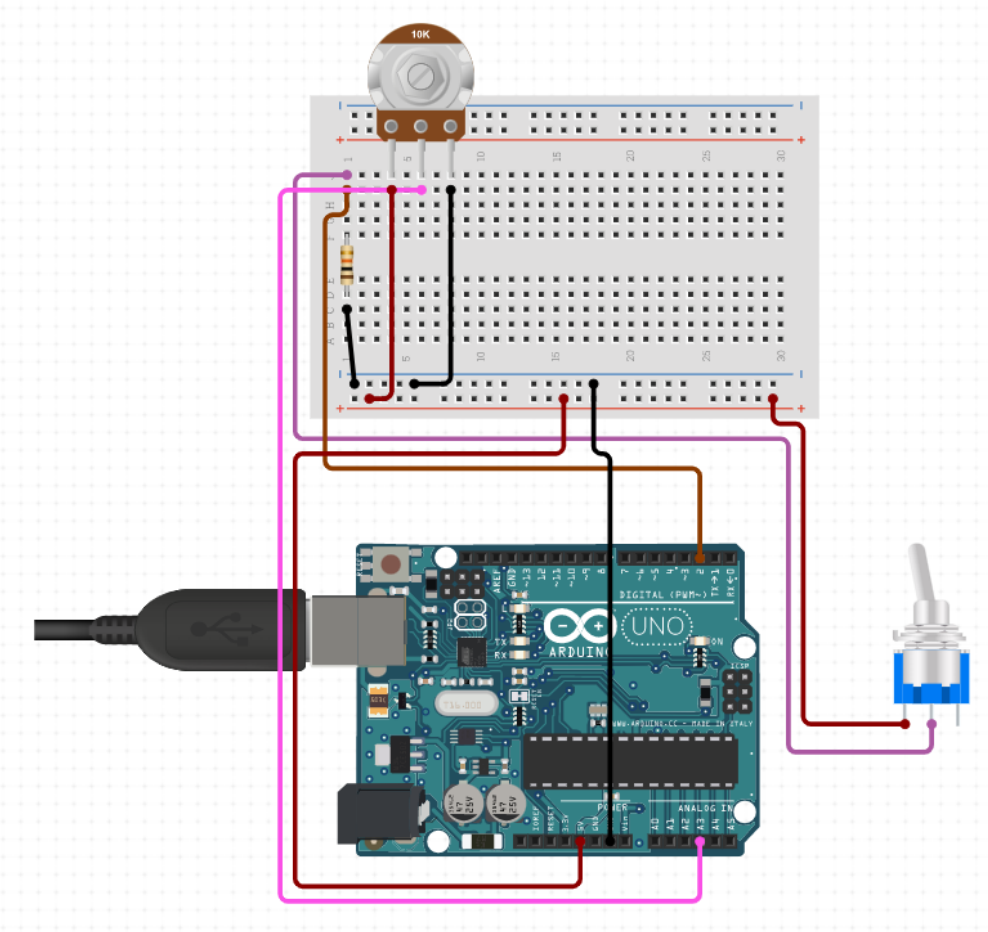

Figure 27 - Arduino circuit

Arduino boards can operate satisfactorily on power that is available from the USB port. It provides 5V DC voltage and can be sourced from the port from a PC, wall socket adapter or portable power bank. Beside power supply purpose, the implemented system relies on the USB cable also to establish a serial communication between Arduino software and MATLAB. Chapter 5.1 describes which is the serial communication protocol adopted in this project.

5.1. UART serial communication protocol

In order to provide a correct functioning of the ECU, a bi-directional flow of data must be realized. As previously explained, data gathered by MATLAB related to sensors must be sent to Arduino in first place, then Arduino must return engine mode and the first injection time assessment (open-loop injection time); MATLAB will gather this data and evaluate, on the basis of this injection time, the error function and the optimal injection time, which in turn must be sent back to Arduino so that the simulated EGO sensor may evaluate the closed-loop injection time, which, again, will be sent back to MATLAB, which will provide significative plots of all the entities assessed.

In general, electrical engineering community decided to standardize electronics around three communication protocols to ensure device compatibility:

1. UART Communication Protocol;
2. SPI Communication Protocol;
3. I^2C Communication Protocol.

A brief discussion will be done regarding the first communication protocol (UART) only, as soon as this is the one used in this project to establish a communication between MATLAB and Arduino.

Acronym of "Universal Asynchronous Receiver/Transmitter" Communication protocol, UART is a form of serial communication where data are transmitted as sequential bits. The wiring involved with setting up UART communication is simple: one line is devoted to transmitting data (TX), while another one is for receiving data (RX).

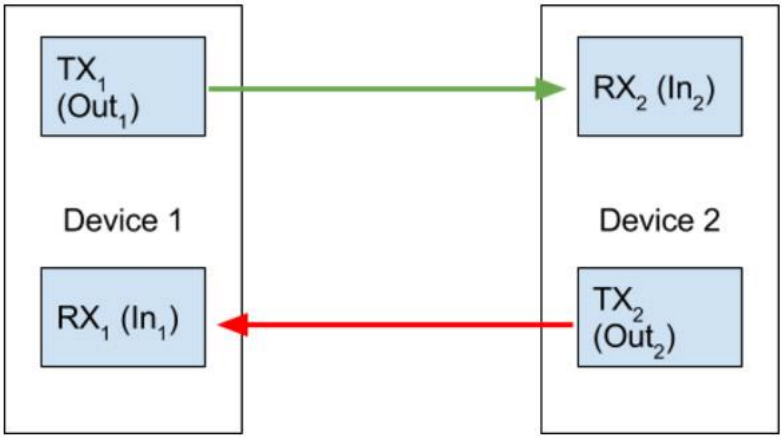

Figure 28 - UART communication protocol scheme [45]

The term UART actually refers to the onboard hardware that manages message packets and transmit serial data. On Arduino Uno there is one serial port devoted to communication with the computer the Arduino is connected to. On this board, USB (Universal Serial Bus) is broken out through onboard hardware into two digital pins, GPIO 0 and GPIO 1, which can be used whenever the project involves serial communication with electronic components other than the computer, as in this case.

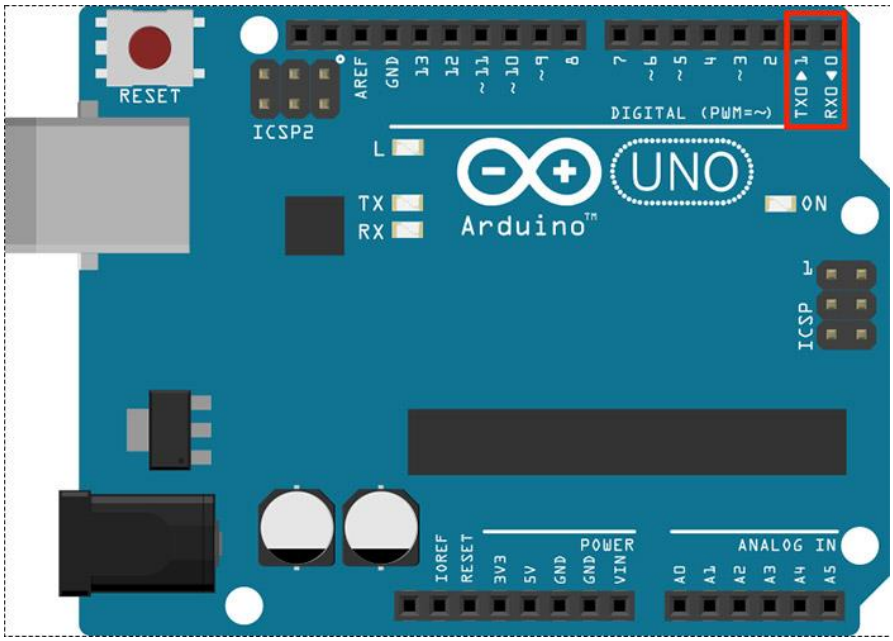

Figure 29 - Arduino Uno GPIO O and GPIO 1 pins [46]

UART is called asynchronous because the communication does not depend on a synchronized clock signal between the two devices attempting to communicate with each other. Because the communication speed is not defined via this steady signal, the sender device cannot be sure that the receiving device obtains the correct data; therefore, the devices decompose data messages into fixed-size pieces, to ensure that data received is the same as the data sent.

A UART data packet is depicted in Figure 30.

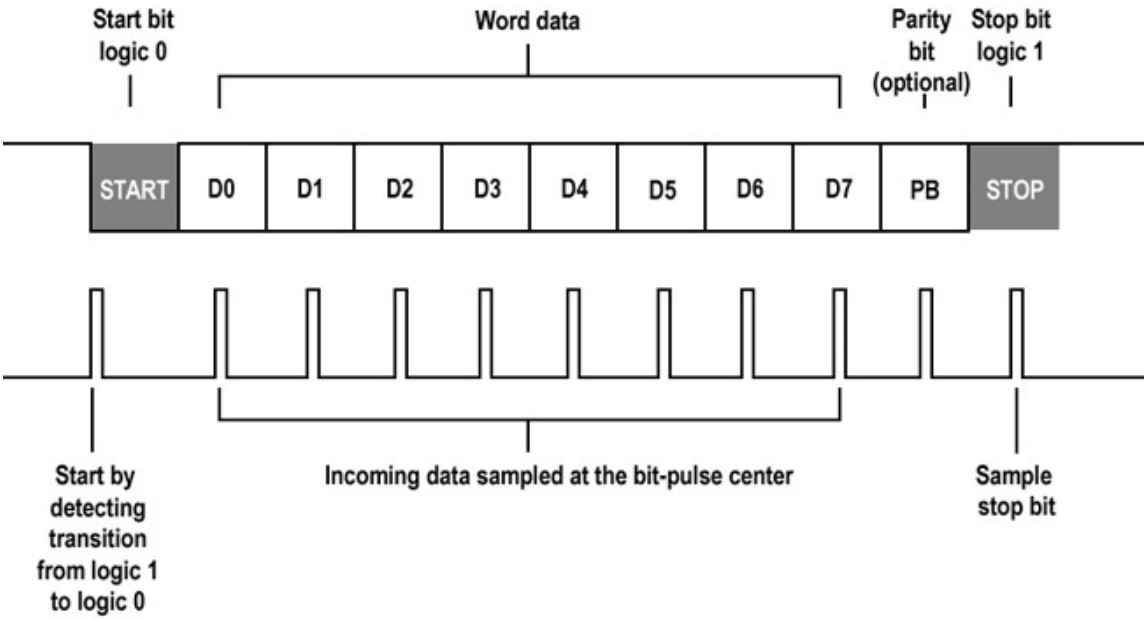

Figure 30 - UART data packet scheme [47]

Devices that communicate via UART send packets of pre-defined size that contain additional information regarding the start and the end of the message and confirmation of whether the message was received correctly. For example, to begin communication, the transmitting device switches from high voltage to low (transition from logic 1 to logic 0), indicating the start of a data packet. Long-term, this means that UART is slower compared to a synchronized form of communication, because only a portion of the data transmitted is for the device's applications.

When using UART serial protocol, the user does not have to deal with communication at the bit level, because the platform often provides higher level software libraries: in Arduino platform, users can use the `Serial` and `SoftwareSerial` libraries to implement UART communication.

In Table 4 [48], a brief C++ reference for Arduino `Serial` and `SoftwareSerial` inizialization and use is proposed.

Serial and SoftwareSerial Method	Purpose	Code	Explanation
Constructor (SoftwareSerial only)	Define the GPIO pins that will serve as the UART RX and TX lines	`SoftwareSerial comms (2,3);`	Defines a serial connection with RX line on GPIO 2 and TX line on GPIO 3.
begin	Define the baud rate (transmission speed) for the serial connection in range 4800 to 115200	`Serial.begin(9600);`	Communication on the serial port will occur at 9600 baud.
print	Write byte data converted into human-readable characters over serial connection	`Serial.println("Hello World");`	Writes bytes equivalent to Hello Work on the serial port.
write	Write raw byte data over the serial connection	`Serial.write(45);`	Writes byte with value 45.
available	Evaluates to true when data is available over the serial connection	`if (Serial.available());`	Enter if statement if there is data available to read over the serial connection

read	Read data from the serial connection	`Serial.read();`	Reads from serial connection.

Table 4 - Serial and SoftwareSerial initialization

One important aspect to take into account when dealing with UART communication is that it is designed for communication between only two devices at a time. Because the protocol only sends bits indicating the start of a message, the message content and the end of the message, there is no method of differentiating multiple transmitting and receiving devices on the same line. If more than one device attempts to transmit data on the same line, bus contention occurs, and the receiving devices will most likely receive unusable data.

Furthermore, UART is half-duplex, which means that even though communication can occur bidirectionally, both devices cannot transmit data to each other at the same time.

This protocol uses a voltage equal to 5V for logic 1 (high), while a voltage of 0V for logic 0 (low). Every message sent by the UART is in the form of 1 byte.
The first step for opening a communication between Arduino and every other device, in this context the computer with MATLAB, is to implement on Arduino IDE, the function `Serial.begin(BaudRate)`, which is a part of the serial object in Arduino and must be inserted inside the `void setup ()`, as depicted in Figure 31, which represent the set of function which are implemented by the board once, at the beginning of the program. This function tells the serial object to perform initialization steps and set the desired baud rate, which indicates the data rate in bit per seconds. The default baud rate in Arduino is 9600 bit per seconds, but in this case 115200 bps were fixed.

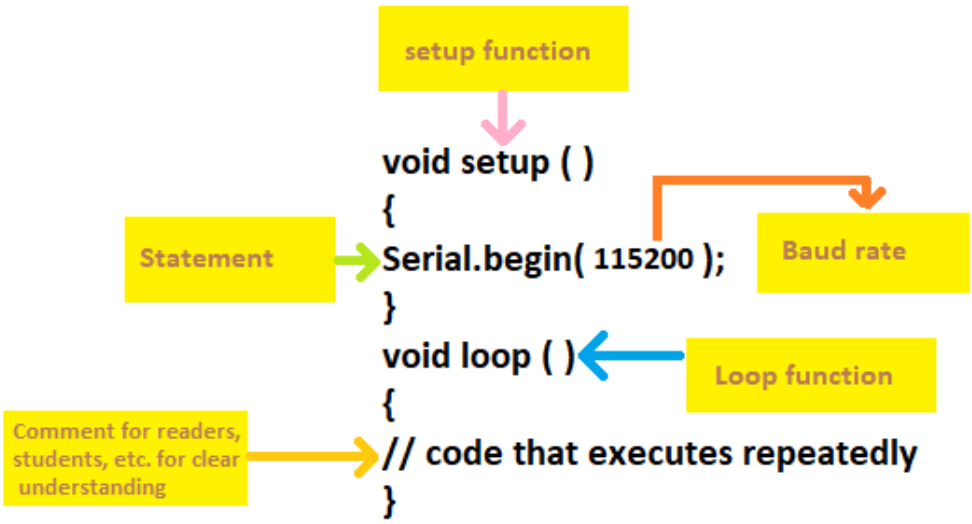

Figure 31 - Arduino IDE setup and loop functions

On the other hand, in MATLAB, to open a serial communication, it is necessary to define, by means of `serial` function (which will be substitute in future MATLAB release by `serialport` object function), a serial port object associated with a specific port (here, "COM3"), as depicted in Figure 32.

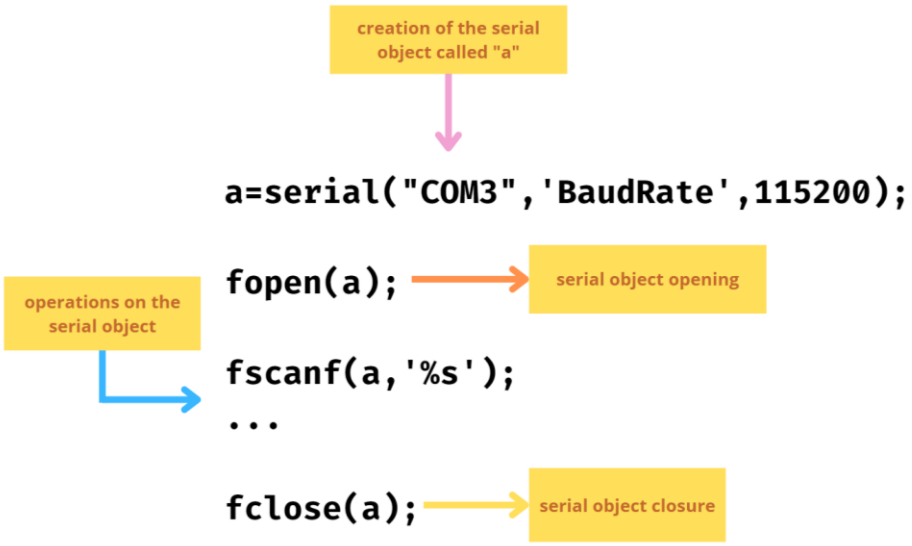

Figure 32 - MATLAB serial object setup and functions

COM (i.e., “Communication port”) is the name of the serial port interface on computers. Generally, it can refer both to physical ports and to emulated ports, such as the ones created by Bluetooth or by, as in this case, USB adapters.

5.2. Logical blocks description

With reference to Figure 25 and Figure 26, which give a general overview of the logic implemented by the prototype, it is worth describing most significative blocks, from the sensors' simulation to the state machine and the injection times calculation.

5.2.1 Sensors' virtualization on MATLAB

As previously mentioned, rpm sensor, coolant medium and EGO temperature sensors and MAF sensors have been virtualized in MATLAB.
All these sensors have been simulated by means of transient functions, as:

$$y(j) = (1 - \alpha) * y(j-1) + \alpha * y_{asymptotic} \quad (22)$$

Where $y(j)$ represents the sensor value at j cycle and α an arbitrarily small constant which is less than 1 and defines the time the function takes to reach the asymptotic value $y_{asymptotic}$ (the largest the value α the shortest the time it takes to approach $y_{asymptotic}$). α was not kept equal for every sensor so that to better simulate a behaviour as close as possible to the reality.

For temperature sensors, the coolant medium one and the EGO one, $y_{asymptotic}$ values have been chosen to be:

Coolant medium temperature sensor	$y_{asymptotic} = 200\ °C$
EGO temperature sensor	$y_{asymptotic} = 600\ °C$

As these values resulted to be consistent with real case maximum conditions.

For what concerns RPM sensor, the function was piecewise defined, to give the virtualization consistency to reality.
Whenever engine mode information sent by Arduino to MATLAB reports that the engine has not

yet reached closed-loop state (conditions to switch from one state to the other will be described in Chapter 5.2.3), RPM function is imposed to be:

$$RPM(j) = (1 - \alpha) * RPM(j - 1) + \alpha * 800 \quad (23)$$

Where 800 rpm has been chosen as the asymptotic value to simulate the situation where the driver has turned the engine ON, but has not pressed the accelerator yet (i.e., TPS information is still not considered significant, which will be done, on the contrary, after closed-loop mode is reached, for sake of simplicity).
As soon as the mode becomes equal to closed-loop, the system is set so that it starts storing potentiometer information from Arduino and the speed becomes dependent on its voltage value, as:

$$RPM(j) = (1 - \alpha) * RPM(j - 1) + \alpha * potentiometer(j) \quad (24)$$

For what concerns MAF sensor, consideration made are similar to the ones related to the speed sensor. Also in this case, the sensor has been virtualized by means of a transient function, where the dependency is set to be on speed, as follows:

$$MAF(j) = (1 - \alpha) * MAF(j - 1) + \alpha * speed(j - 1) \quad (25)$$

Of course, in actual conditions, the air-flow aspirated in the manifold does not directly depend on the speed of the crankshaft, but, for sake of simplicity, it has been considered that this sensor behaviour may reasonably permit students to appreciate the core logic of the ECU's working principles.
As soon as the engine is sensed to be on OFF mode, speed and MAF sensors are set to equal 0, while potentiometer information coming from Arduino is by-passed and the potentiometer function is set equal to zero as well. For what concerns temperature sensors, on the contrary, it is clear that in actual conditions the temperature does not switch to ambient one immediately. For this reason, also in this case, a transient behaviour is foreseen, by imposing:

$$temperature(j) = (1 - \alpha) * temperature(j - 1) + \alpha * 25 \quad (26)$$

Where 25°C is the ambient temperature. This equation has been imposed for both coolant medium temperature sensor and EGO temperature sensor.

5.2.2 Sensors' data reading on Arduino

As in MATLAB sensors have been virtualized by means of code as previously explained, in Arduino environment, a slide switch and a potentiometer have been physically assembled on the board and their signals detected and stored so that to be used, together with data coming from MATLAB, to implement the state machine.
As previously explained, being the logical output of a digital pin equal to 0 or to 1, this has been used to connect the slide switch, which information is just that related to the ON/OFF state of the engine. The reading of the value of this sensor by Arduino is very straightforward, as soon as Arduino libraries also include specific reading function for digital signals.
On the other hand, the potentiometer voltage is an analog input, which needs to be converted into digital before acquisition. For this purpose, the Analog-to-Digital converter embedded in the microcontroller is used. The A/D converter is controlled via software by configuring the I/O pin as an analog input and then by using the suitable library functions for the ADC reading.
As previously seen in Chapter 4, analog pins can map input voltages by using integers in a range going from 0 to 1023; this means that, to retrieve voltage information, it is necessary to implement a simple proportion:

$$potentiometer\ voltage = \frac{read\ value}{1023} * 5 \quad (27)$$

As soon as this equation is implemented, the information coming from the potentiometer is ready to be used by Arduino software to implement the state machine.

5.2.3 State Machine implementation

As soon as, once per second, sensors data are sent by MATLAB on the serial and read and stored by Arduino, the latter can start implementing the function devoted to simulate the state machine, after having gathered data coming from the potentiometer as well (only if the slide switch is on ON mode, of course).
The state machine is a mathematical model of computation: it consists in an abstract machine that can be exactly one of a finite number of states at any given time. It can switch from one state to another in response of some inputs and it is defined by a list of states, the initial state and the inputs that trigger each transition. In this case, possible states are: cranking (defined also as initial state), warm-up, open-loop, closed-loop, hard acceleration, deceleration and idle. Inputs which trigger transitions are the sensors data.
ECU control mode algorithm is here reported, in Figure 33, with switch conditions, to better understand which is the sequence of actions implemented by the Arduino microprocessor while simulating the state machine:

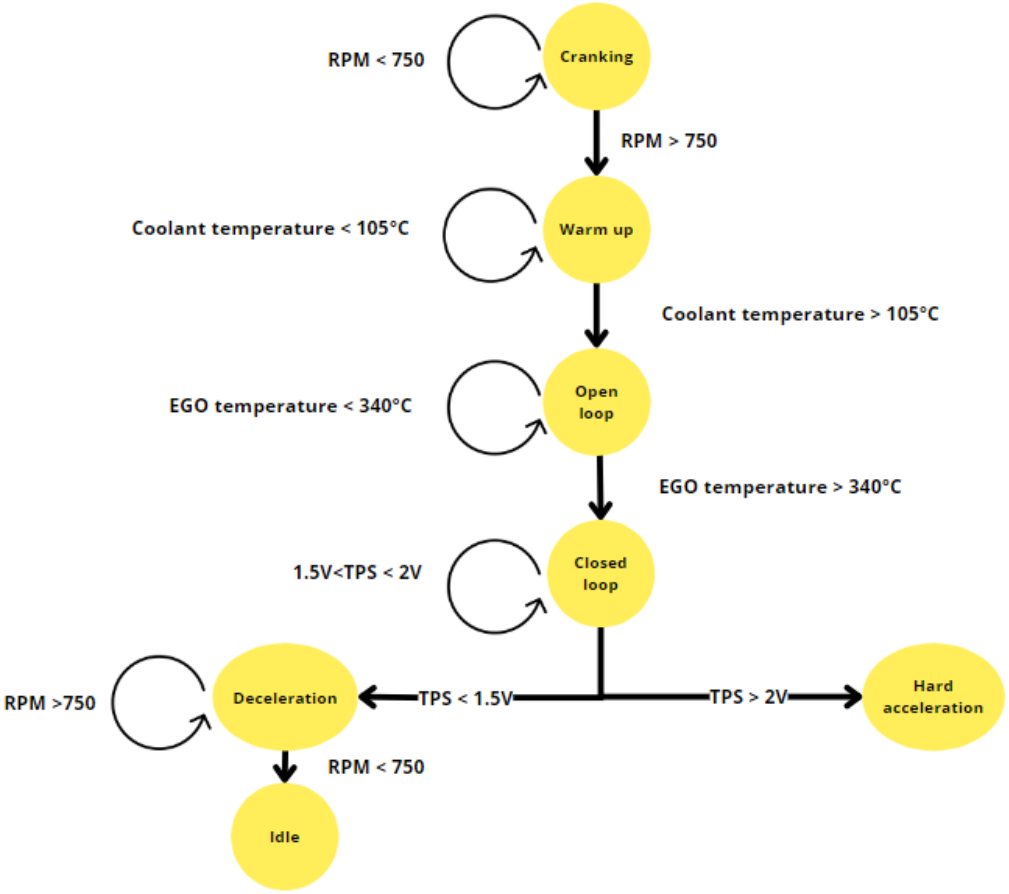

Figure 33 - State machine control algorithm

Summarizing:

- If the engine is in CRANKING mode, to switch to WARM-UP it is necessary that the speed of the crankshaft has overcame a specific threshold: here, 750 rpm;
- If the engine is in WARM-UP mode, to switch to OPEN-LOOP it is necessary that the temperature of the coolant medium has overcame a specific threshold: here, 105 °C;
- If the engine is in OPEN-LOOP mode, to switch to CLOSED-LOOP it is necessary that the EGO temperature sensor has overcame a specific threshold: here, 340 °C;
- If the engine is in CLOSED-LOOP mode, to switch to HARD ACCELERATION it is necessary that the voltage of the potentiometer has overcame a specific threshold: here, 2V;
- If the engine is in CLOSED-LOOP mode, to switch to DECELERATION it is necessary that the voltage of the potentiometer has become less than a specific threshold: here, 1.5V;
- If the engine is in DECELERATION mode, to switch to IDLE it is necessary that the speed of the crankshaft returns to be less than a specific threshold: again, 750 rpm;
- Whenever CLOSED-LOOP is reached, state may eventually be switched to HARD ACCELERATION (if voltage is larger than 2V) or DECELERATION (if voltage is less than 1.5V). If voltage is kept between 1.5V and 2V, the engine remains in CLOSED-LOOP mode.

Every threshold value has here been chosen arbitrarily. Specifically, the condition to switch from cranking mode to warm-up mode, related to the speed, has been chosen by considering that a typical cranking rpm window can be from 50 to 400 rpm, which means that 750 rpm may be a reasonable value to allow the engine to enter properly the warm-up phase.
For what concerns temperature sensors, it seems reasonable to consider a nominal temperature for coolant medium about 200 °C, which means that above more or less 105°C the warm-up phase can be considered over. Same considerations stand for EGO temperature sensor: as soon as it is reasonable to consider that temperatures of 500-700°C are produced in the exhaust gases from diesel-cycle engines at 100% load to 200-300°C with no load[49], 340°C seem to be a fair threshold to switch from open-loop to closed-loop.
Regarding potentiometer voltage, i.e., TPS simulated signal, values have been chosen by considering a voltage window going from 0V to 5V. As soon as hard acceleration and deceleration are deviations with respect to closed-loop injection time, which means that in this case the correct fuel amount to be injected inside the cylinders is assessed on the basis of the power demand, the system is set to leave closed-loop whenever the voltage is sensed to be higher than 2V to switch

to hard acceleration mode and to deceleration mode whenever it is sensed to be less than 1.5, so that to leave the window from 1.5V to 2V for actual closed-loop mode to be implemented. Whenever deceleration mode is reached, if the crankshaft speed is reduced under the threshold of 750 rpm, the engine may be considered to be in idle mode. At this point, the engine may be turned OFF and then ON again (which means the state machine will start over from cranking) or be kept ON and in this case, according to TPS voltage, the engine mode can be switched from idle mode to closed-loop and eventually hard acceleration again.

The information related to the engine mode is sent from Arduino to MATLAB by means of the serial bus and here is used in order to properly implement sensors virtualization (for example, as previously described, speed function is piecewise defined on the basis of the engine mode information coming from Arduino) and to actually display the engine mode (UART communication, as previously said, does not permit to transmit data on the same line, which means that it is not possible to open the Arduino serial monitor to visualize outputs; on the contrary, they can be displayed on MATLAB only, as far as the serial object is open).

5.2.4 Fuel injection times calculation

As depicted in Figure 26, which described the logic flow followed by the prototype to assess injection times, data related to two sensors, i.e., rpm sensor and MAF sensor, are used not only to properly define which is the current engine mode, but also for fuel injection control purposes. Equation (5) describes how to calculate the open-loop injection time, which is the information used by the ECU to define the proper amount of fuel to inject inside the cylinders as far as EGO sensor temperature has not yet reached the nominal value.
Data necessary to assess this value are:

- Air flow rate (MAF signal);
- Crankshaft rotational speed (RPM signal);
- R_f (fuel injection rate);
- Air/fuel stoichiometric ratio.

First two values, as previously seen, are sent on the serial from MATLAB and stored in Arduino in proper arrays.
For what concerns R_f, a significative value has been chosen by considering a study [50]carried on considering a direct injection engine relying on a “Siemens Deka 4” injector, with 3 bars injection

pressure and 4.3 g/s delivery rate. Finally, being this project based on the simulation of a gasoline engine control ECU, the air/fuel stoichiometric ratio is equal to 14.7.

As soon as open-loop injection time is assessed on the basis of the data described, its value is sent on the serial by Arduino to MATLAB. Here, it is stored and used to assess the optimal injection time as the sum of the open-loop injection time and an error function, assessed as:

$$error\ function(j) = (1 - \alpha) * error\ function(j - 1) * \alpha * r(j) \quad (28)$$

Where $r(j)$ corresponds to a function which returns a random number within a specific range and it is defined as:

$$r(j) = (max - min) *. rand(1) + min \quad (29)$$

The necessity to adopt a transient function also for defining the error, as it has been done for sensors, derives from the willing to obtain a behaviour which is sufficiently smooth, so that changes can be fully appreciated and stored in optimal injection time calculation. As soon as optimal injection time has been calculated, as:

$$T_{inj,OPTIMAL}(j) = T_{inj,OPEN\ LOOP}(j) + T_{ERROR} \quad (30)$$

This value is sent to Arduino, where the EGO sensor is virtualized. Of course, the actual working principle of the EGO sensor involves the detection of the oxygen level in the exhaust gas, so that to detect whether the mixture is lean or rich. In this case, due to the impossibility to retrieve data from real exhaust gases, the EGO variable, defined in Eq. (7) is defined on the basis of the comparison between the optimal injection time and the closed-loop one; specifically, it has been defined that:

$T_{inj,CLOSED\ LOOP}(j) < T_{inj,OPTIMAL}(j)$	$EGO(j) = -1$
$T_{inj,CLOSED\ LOOP}(j) > T_{inj,OPTIMAL}(j)$	$EGO(j) = +1$

As described by Eq. (8), a corrective factor $C_L(n)$ for the n-th cycle is evaluated on the basis of the sum of the $EGO(j)$ values stored for all the previous cycles, until the $(n-1)$-th (integral control) and of the integral factor k_I; this latter is less than 0 and has been chosen arbitrarily small. $C_L(n)$ is then inserted inside Eq. (6) to assess the closed-loop injection time.

This value is then sent to MATLAB, which plots optimal injection time, open-loop injection time and closed-loop injection time.

5.3. Significative plots

This sub-chapter is intended to provide several significative plots, which will be shortly commented, so that to allow a clear visualization of the simulation outputs in different operating conditions and to provide a better understanding of the control operations.

5.3.1 Sensors plots

Here, relevant signals from real and virtual sensors are plotted and, for every iteration (which takes place once per second) corresponding engine modes are displayed.

Figure 34 shows sensors behaviour corresponding to the first two engine modes, i.e., cranking and warm up.

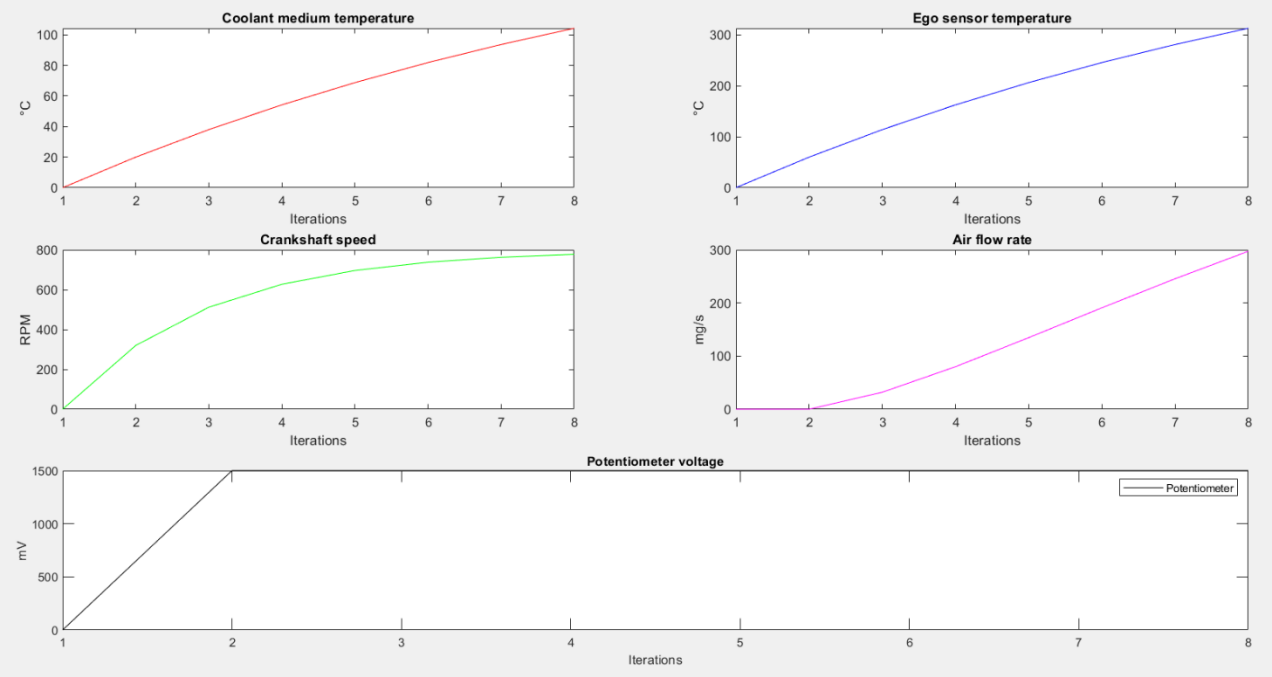

Figure 34 - Sensors plots related to cranking and warm-up modes

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
```

As it possible to notice:

- Potentiometer function (black): here kept to 1.5V; in the first phases has no impact at all on speed and MAF;
- Speed function (green): approaching the asymptotic value of 800 rpm;
- MAF sensor (pink) function: moving towards the speed function;
- Cooling temperature sensor (red): moving towards the asymptotic value of 200°C;
- EGO temperature sensor (blue): moving towards the asymptotic value of 600°C.

Figure 35 shows sensors behaviour when the Closed-Loop mode is reached.

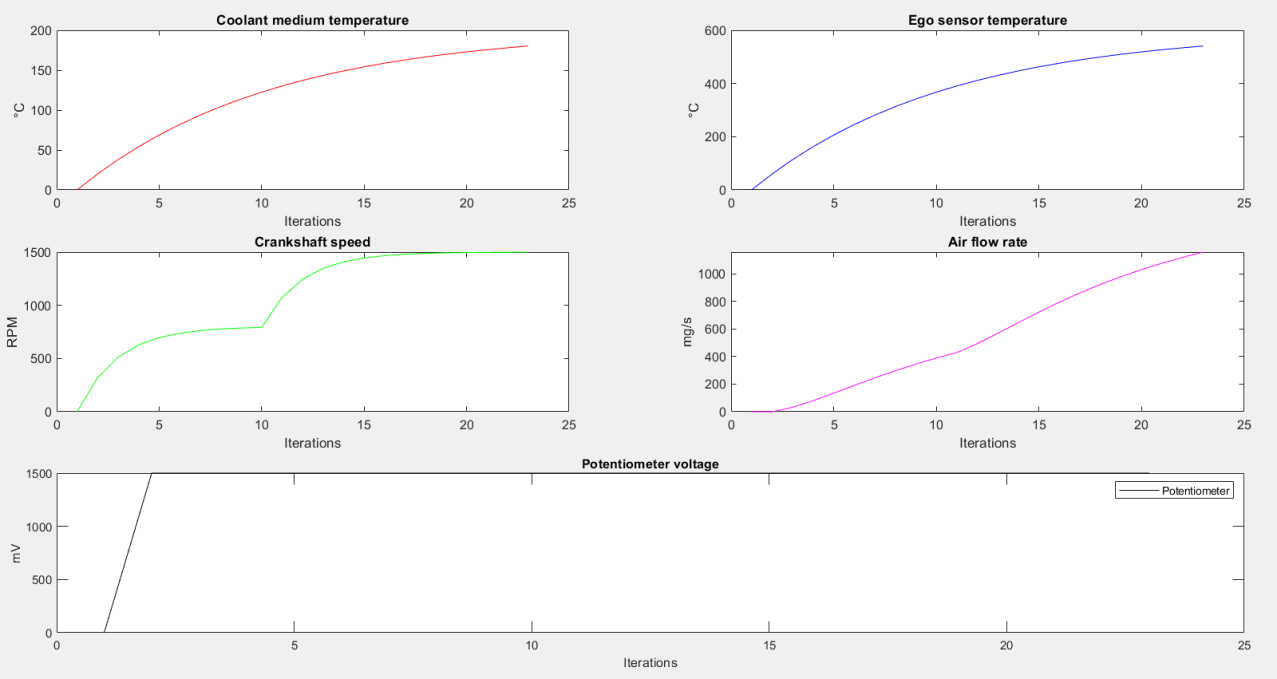

Figure 35 - Sensors plots related to cranking, warm-up open-loop and closed-loop modes

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: WARM UP
Iteration number 9: OPEN LOOP
Iteration number 10: CLOSED LOOP
Iteration number 11: CLOSED LOOP
Iteration number 12: CLOSED LOOP
Iteration number 13: CLOSED LOOP
Iteration number 14: CLOSED LOOP
Iteration number 15: CLOSED LOOP
Iteration number 16: CLOSED LOOP
Iteration number 17: CLOSED LOOP
Iteration number 18: CLOSED LOOP
Iteration number 19: CLOSED LOOP
Iteration number 20: CLOSED LOOP
Iteration number 21: CLOSED LOOP
Iteration number 22: CLOSED LOOP
Iteration number 23: CLOSED LOOP
Iteration number 24: CLOSED LOOP
Iteration number 25: CLOSED LOOP
```

- Potentiometer function (black): here kept to 1.5V; in the first phases (cranking, warm up and open-loop) has no impact at all on speed and MAF; at 10th iteration, when closed-loop is reached, potentiometer is always kept to 1.5V, impacting speed function;
- Speed function (green): after closed-loop is reached at 10th iteration, starts increasing, as potentiometer is kept to 1.5V.
- MAF sensor (pink) function: still moving towards the speed function; as MAF is related to `speed (j-1)` value, it will change its slope starting by 11th iteration.
- Cooling temperature sensor (red): moving towards the asymptotic value of 200°C;
- EGO temperature sensor (blue): moving towards the asymptotic value of 600°C.

Figure 36 shows sensors behaviour when Deceleration and Idle modes are reached.

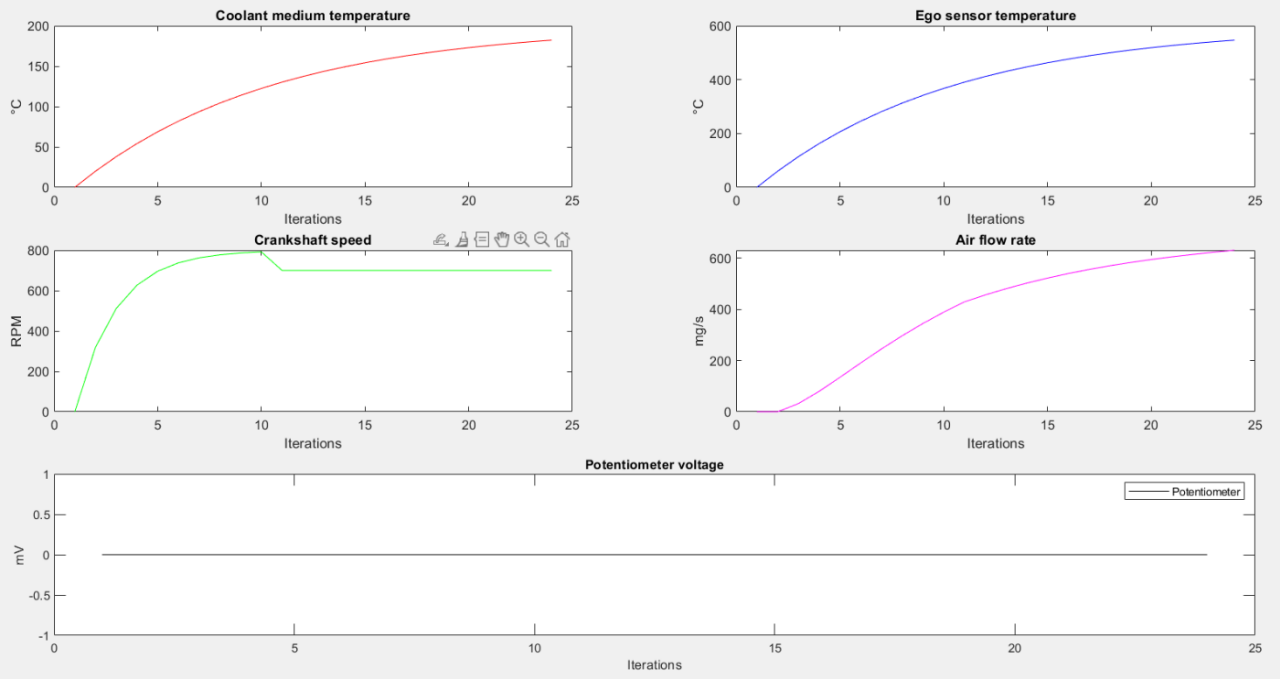

Figure 36 - Sensors plots related to cranking, warm-up open-loop, closed-loop, deceleration and idle modes

```
Iteration number 9: CLOSED LOOP
Iteration number 10: DECELERATION
Iteration number 11: IDLE
Iteration number 12: IDLE
Iteration number 13: IDLE
Iteration number 14: IDLE
Iteration number 15: IDLE
Iteration number 16: IDLE
Iteration number 17: IDLE
Iteration number 18: IDLE
Iteration number 19: IDLE
Iteration number 20: IDLE
Iteration number 21: IDLE
Iteration number 22: IDLE
Iteration number 23: IDLE
```

- Potentiometer function (black): kept to zero;
- Speed function (green): after closed-loop is reached at 9th iteration, starts decreasing, as potentiometer is kept to 0V.
- MAF sensor (pink) function: approaching `speed (j-1)` value.
- Cooling temperature sensor (red): approaching the asymptotic value of 200°C;
- EGO temperature sensor (blue): approaching the asymptotic value of 600°C.

Fig. shows sensors behaviour when Hard Acceleration mode is reached.

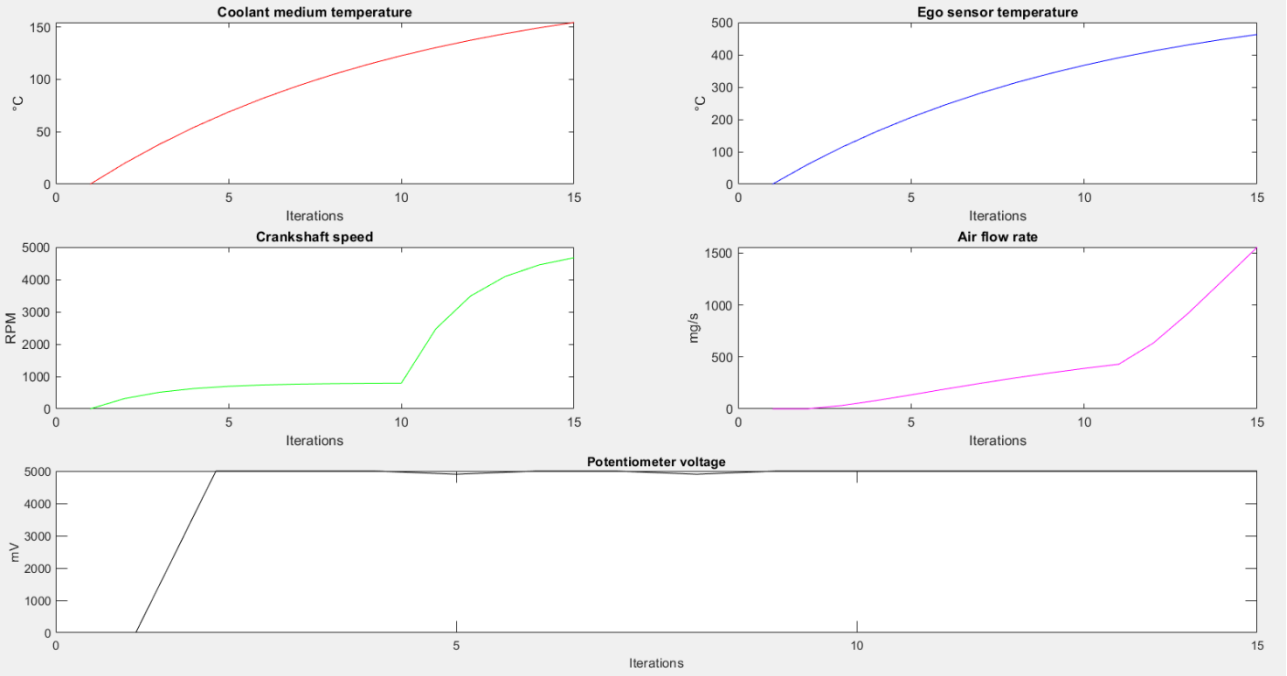

Figure 37- Sensors plots related to cranking, warm-up open-loop, closed-loop and hard acceleration modes

```
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: OPEN LOOP
Iteration number 9: CLOSED LOOP
Iteration number 10: HARD ACCELERATION
Iteration number 11: HARD ACCELERATION
Iteration number 12: HARD ACCELERATION
Iteration number 13: HARD ACCELERATION
Iteration number 14: HARD ACCELERATION
```

- Potentiometer function (back): set to the maximum value, i.e., 5V;
- Speed function (green): after closed-loop is reached at 9th iteration, starts increasing according to potentiometer value;
- MAF sensor (pink) function: moving towards `speed (j-1)` value.
- Cooling temperature sensor (red): approaching the asymptotic value of 200°C;
- EGO temperature sensor (blue): approaching the asymptotic value of 600°C.

Figure 38 allows to appreciate how changes in potentiometer voltage affects speed and in turn MAF sensors and allows the switch among the different engine modes.

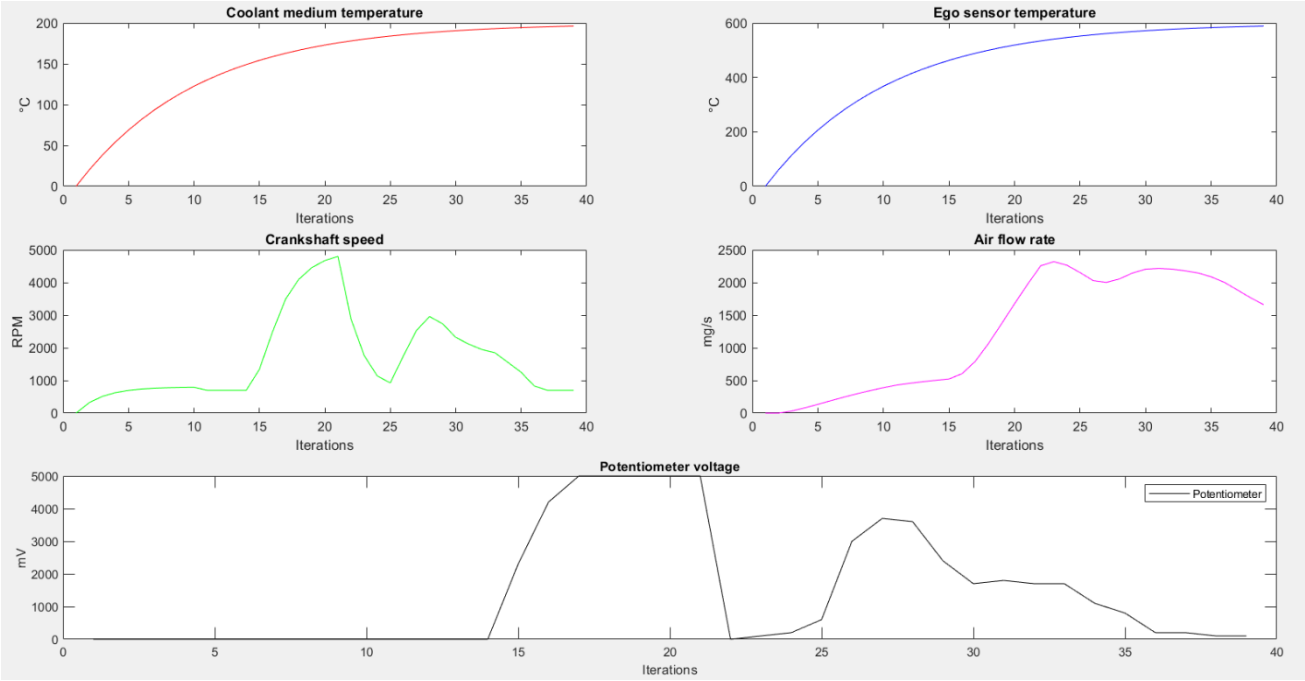

Figure 38 - Sensors plots changing according to potentiometer inputs

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: WARM UP
Iteration number 9: OPEN LOOP
Iteration number 10: CLOSED LOOP
Iteration number 11: DECELERATION
Iteration number 12: IDLE
Iteration number 13: IDLE
Iteration number 14: IDLE
Iteration number 15: CLOSED LOOP
Iteration number 16: HARD ACCELERATION
Iteration number 17: HARD ACCELERATION
Iteration number 18: HARD ACCELERATION
Iteration number 19: HARD ACCELERATION
Iteration number 20: HARD ACCELERATION
Iteration number 21: DECELERATION
Iteration number 22: DECELERATION
Iteration number 23: DECELERATION
Iteration number 24: DECELERATION
```

```
Iteration number 25: HARD ACCELERATION
Iteration number 26: HARD ACCELERATION
Iteration number 27: HARD ACCELERATION
Iteration number 28: HARD ACCELERATION
Iteration number 29: CLOSED LOOP
Iteration number 30: CLOSED LOOP
Iteration number 31: CLOSED LOOP
Iteration number 32: CLOSED LOOP
Iteration number 33: DECELERATION
Iteration number 34: DECELERATION
Iteration number 35: DECELERATION
Iteration number 36: DECELERATION
Iteration number 37: IDLE
Iteration number 38: IDLE
```

As depicted in Figure 38, potentiometer signal is kept to 0V until 13th iteration, but does not affect speed and MAF until closed-loop mode is reached, in iteration 10th. After that, speed starts to

decrease, moving towards 0 rpm. According to speed function, MAF sensor signal changes in scope and starts to decrease more steeply to reach `speed (j-1)` value.

Deceleration and then idle mode is reached.

After this phase, potentiometer voltage is progressively increased, so that to reach closed-loop mode again: speed and MAF increase accordingly. After iteration 15th, potentiometer signal continues to increase, and hard acceleration mode is reached.

Several changes in potentiometer voltage have been done, so that to appreciate how other sensors behave accordingly.

In Figure 39 it is possible to see sensors behaviour as soon as the engine is turned OFF.

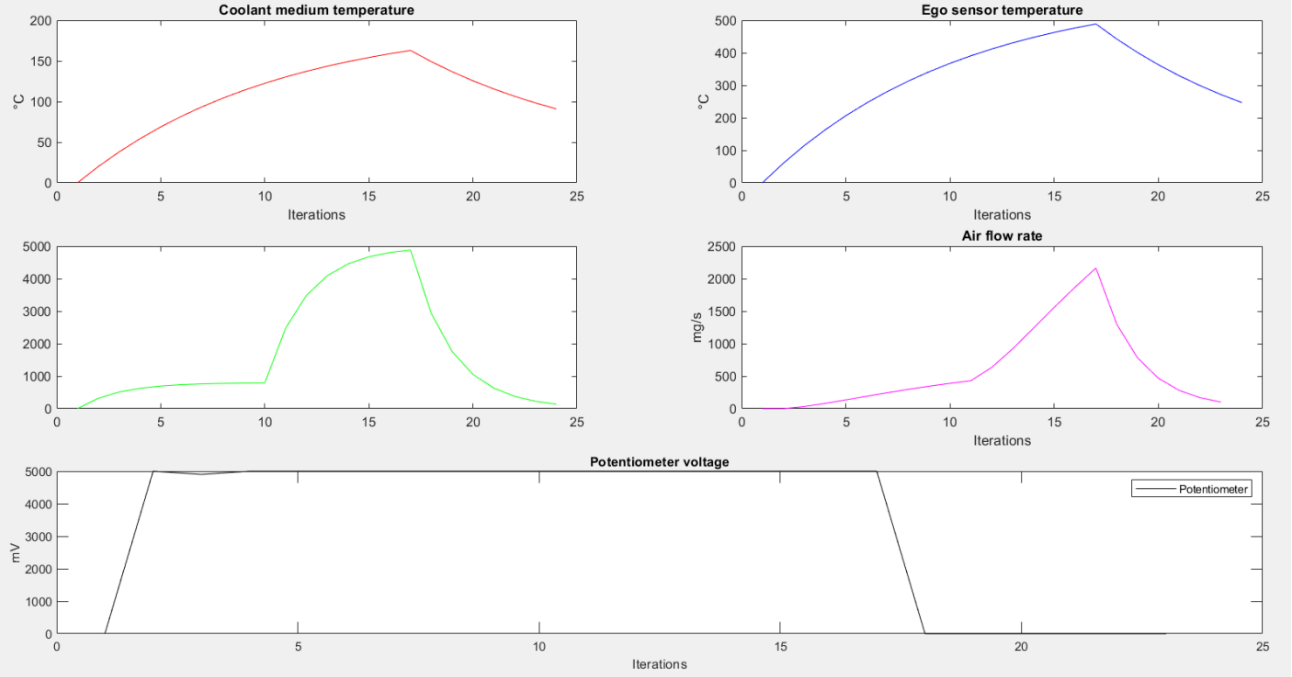

Figure 39 - Sensors plots when engine is turned OFF

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: WARM UP
Iteration number 9: OPEN LOOP
Iteration number 10: CLOSED LOOP
Iteration number 11: HARD ACCELERATION
Iteration number 12: HARD ACCELERATION
Iteration number 13: HARD ACCELERATION
Iteration number 14: HARD ACCELERATION
Iteration number 15: HARD ACCELERATION
Iteration number 16: HARD ACCELERATION
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
```

- Potentiometer function (back): goes to 0V when the engine is turned OFF, in 17th iteration;
- Speed function (green): starts decreasing, in order to reach 0 rpm, when the engine is turned OFF;
- MAF sensor (pink) function: starts decreasing, in order to reach 0 rpm, when the engine is turned OFF;
- Cooling temperature sensor (red): starts decreasing, in order to reach 25 °C, when the engine is turned OFF;
- EGO temperature sensor (blue): starts decreasing, in order to reach 25 °C, when the engine is turned OFF;

Figure 40 shows sensors behaviour whenever the engine is turned ON again, after being turned OFF.

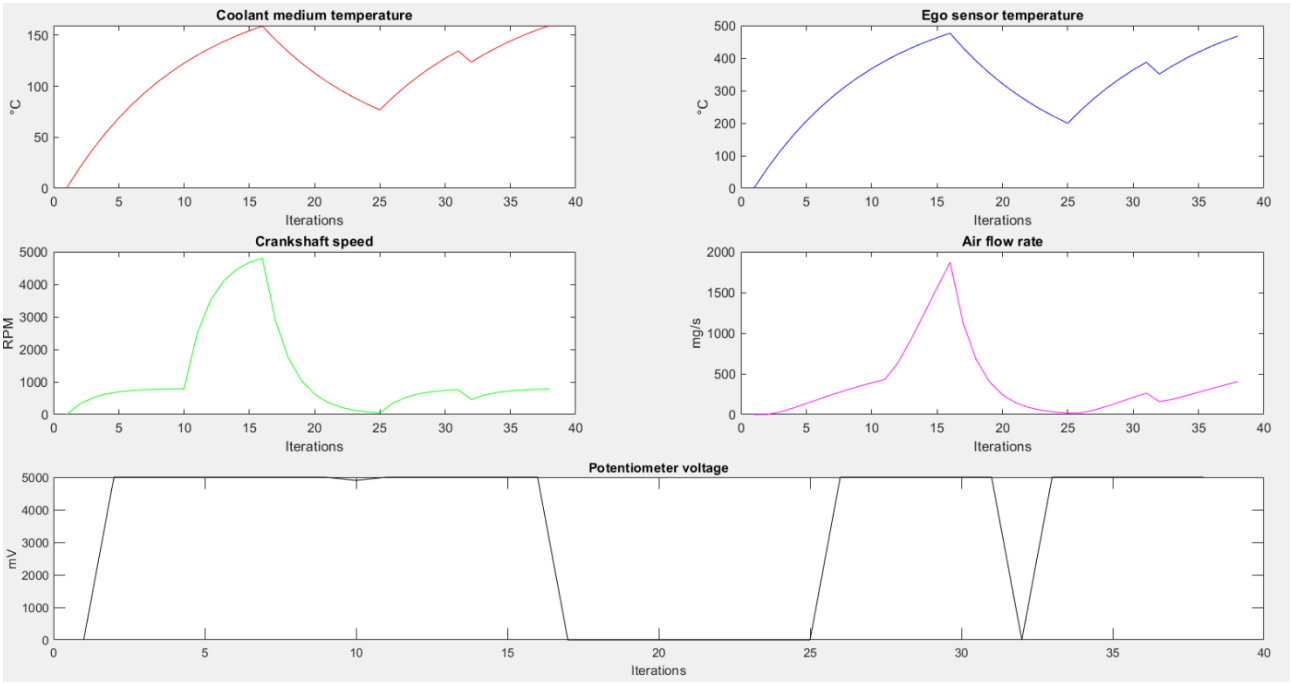

Figure 40 - Sensors plots when engine is turned OFF and then ON again

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: WARM UP
Iteration number 9: OPEN LOOP
Iteration number 10: CLOSED LOOP
Iteration number 11: HARD ACCELERATION
Iteration number 12: HARD ACCELERATION
Iteration number 13: HARD ACCELERATION
Iteration number 14: HARD ACCELERATION
Iteration number 15: HARD ACCELERATION
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
ENGINE OFF
```

```
Iteration number 25: CRANKING
Iteration number 26: CRANKING
Iteration number 27: CRANKING
Iteration number 28: CRANKING
Iteration number 29: CRANKING
Iteration number 30: CRANKING
ENGINE OFF
Iteration number 32: CRANKING
Iteration number 33: CRANKING
Iteration number 34: CRANKING
Iteration number 35: CRANKING
Iteration number 36: WARM UP
Iteration number 37: OPEN LOOP
```

As it is possible to notice, after the engine is turned ON again, sensors resume the previous expected behaviour, by starting from the last value they assumed when the engine was OFF.

5.3.2 Fuel injection times plots

In this sub-chapter, fuel injection times plots are provided, in order to better appreciate control operations.

In order to appreciate the correction implemented by closed-loop time over the open-loop, Figure 41 is proposed, where open-loop time has been kept constant at 300 ms, so that to verify the correct behaviour of the closed-loop one with respect to the optimal time.

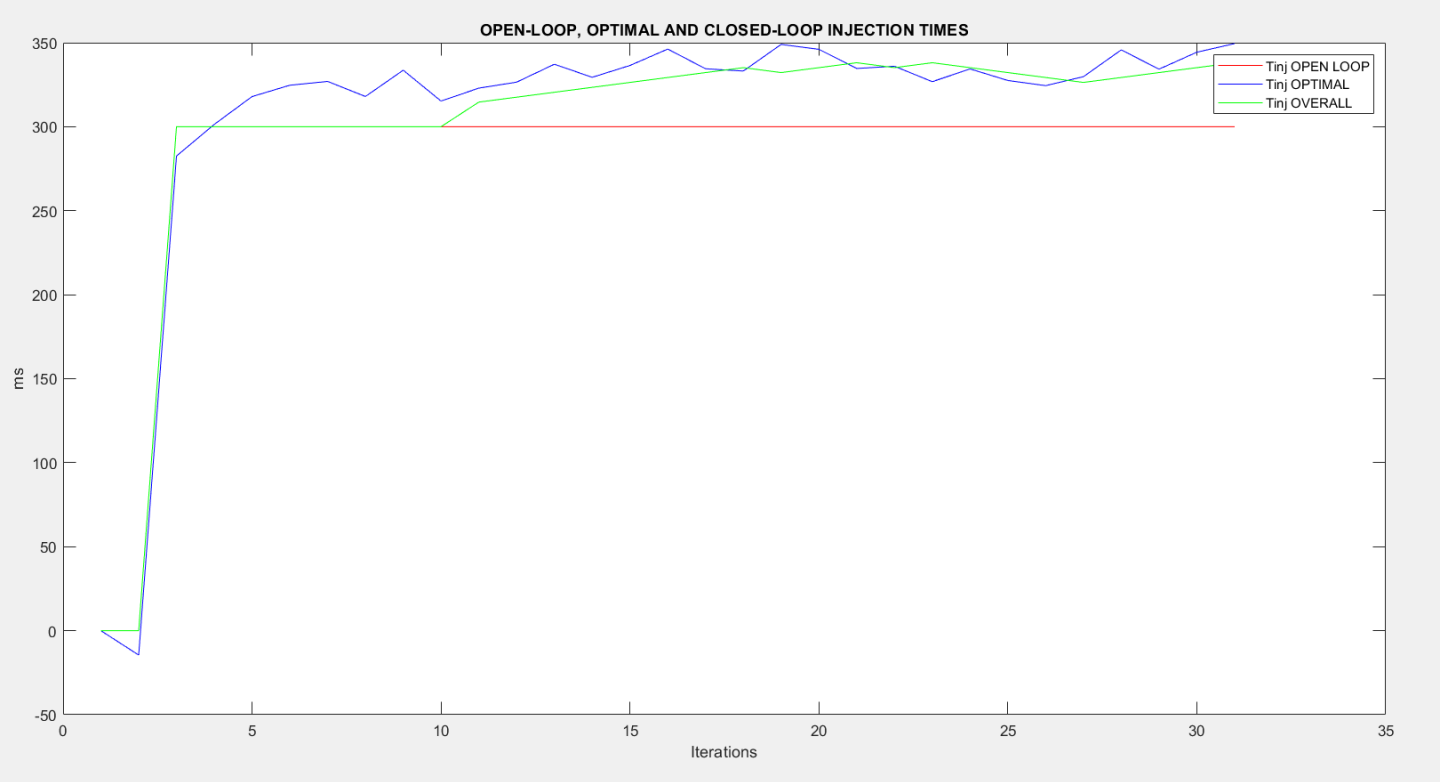

Figure 41 - Injection times when open-loop injection time is kept constant

As it is possible to see, the overall injection time remains constant and equal to open-loop injection as soon as closed-loop mode is reached in 11^{th} iteration. At this point, it becomes equal to closed-loop and starts behaving according to the comparison made between closed-loop and optimal time. As soon as closed-loop injection time results to be less than optimal one, closed-loop time increases; on the other hand, whenever closed-loop time is larger than optimal one, closed-loop time starts decreasing, correcting in this way the amount of fuel to be injected.

Figure 42 are proposed to appreciate control behaviour as soon as open-loop injection time is no more kept constant, as in actual conditions.

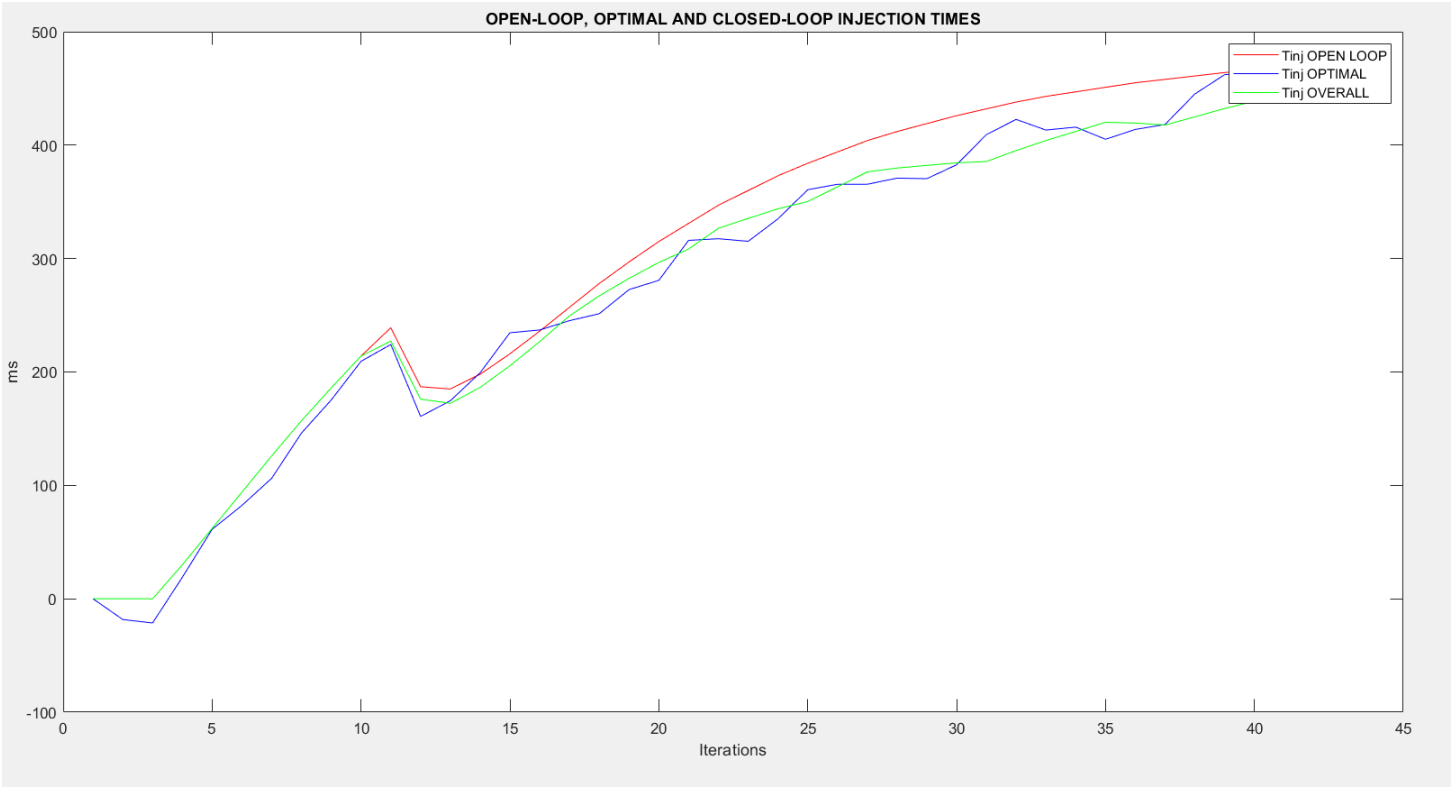

Figure 42 - Injection times with changing open-loop time

```
Iteration number 1: CRANKING
Iteration number 2: CRANKING
Iteration number 3: CRANKING
Iteration number 4: CRANKING
Iteration number 5: CRANKING
Iteration number 6: CRANKING
Iteration number 7: WARM UP
Iteration number 8: WARM UP
Iteration number 9: OPEN LOOP
Iteration number 10: CLOSED LOOP
Iteration number 11: CLOSED LOOP
Iteration number 12: CLOSED LOOP
```

As it is possible to notice from the plot, closed-loop, also in this case, corrects the injection time so that to be more consistent with optimal value.

Figure 43 - Sensors behaviourand Figure 44 are here proposed in order to show how injection time control works for different engine states.

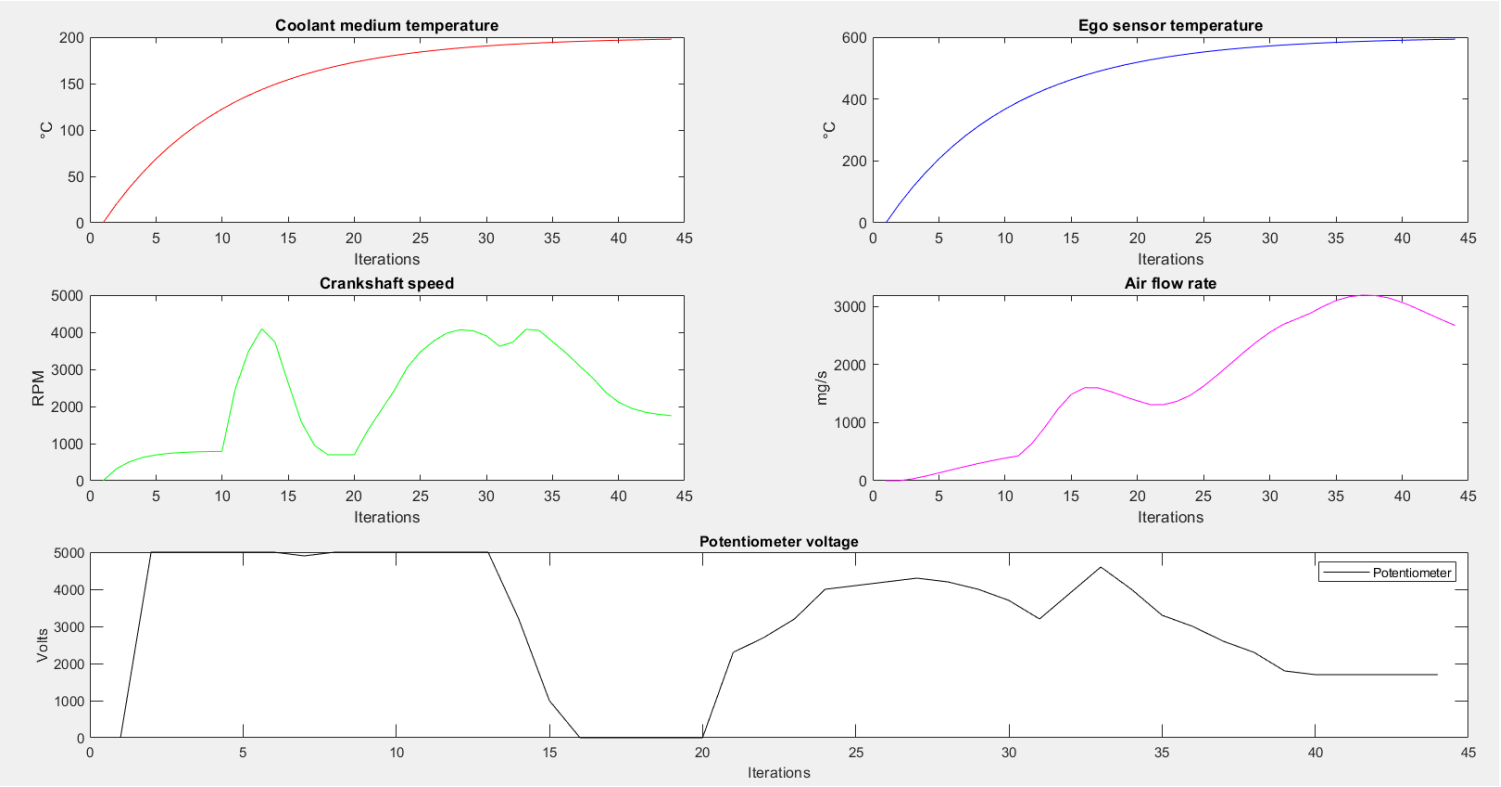

Figure 43 - Sensors behaviour

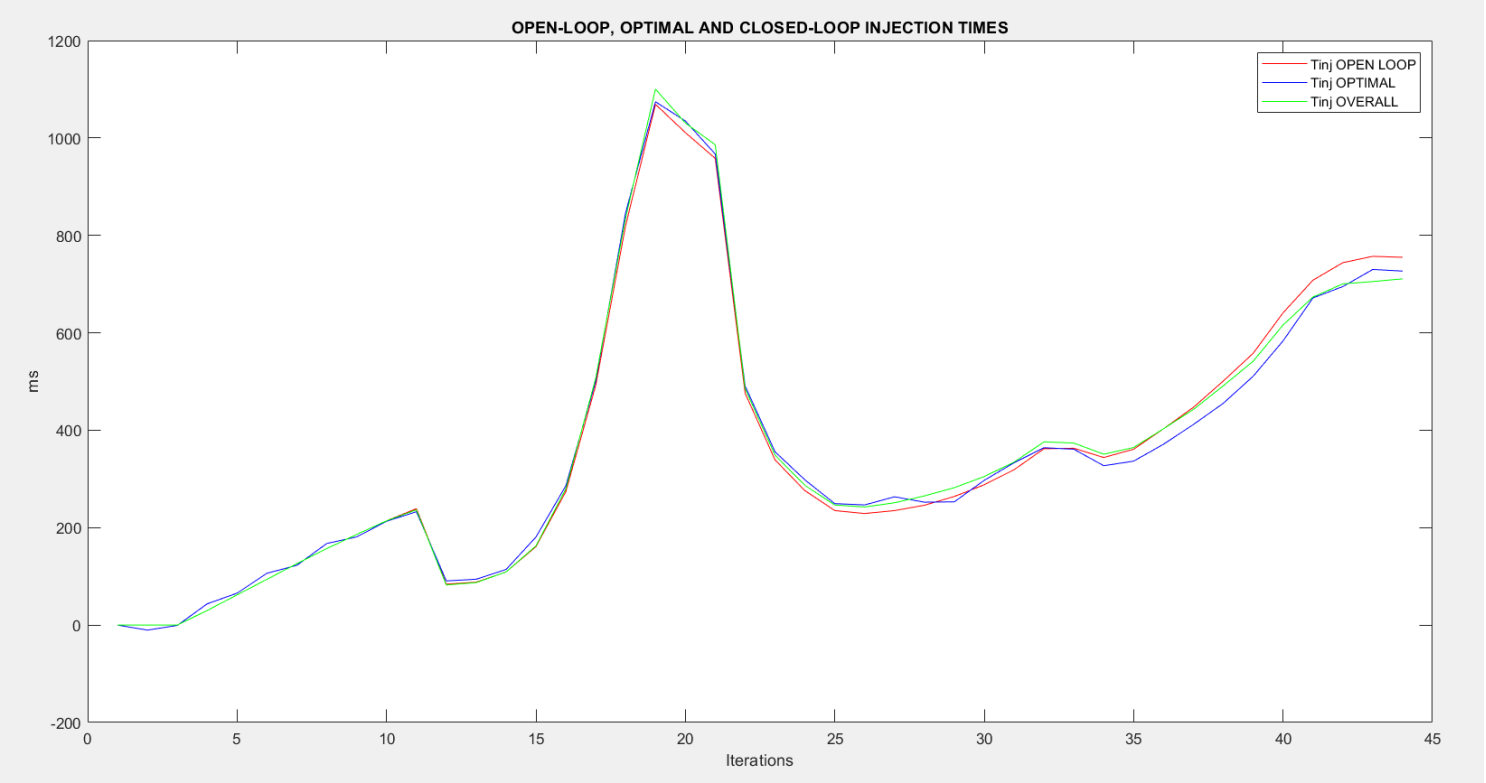

Figure 44 - Injection times

All injection times have been set to go to zero whenever the engine is OFF.

5.4. Code overview

MATLAB and Arduino code are here fully reported, in order to depict the actual C and C++ functions which have been used in designing the prototype.

5.4.1 MATLAB code

```
clear;
close all;
clc;
fclose(instrfindall);

a=serial("COM3",'BaudRate',115200);
fopen(a);

%Initialization
j=2;
temperature(1)=25;
temperature_ego(1)=25;
potentiometer(1)=0;
speed(1)=0;
air_flow_rate(1)=0;
T_error(1)=0;

temperature_last_OFF=0;
temperature_ego_last_OFF=0;
speed_last_OFF=0;
MAF_last_OFF=0;
T_error_last_ON=0;


%Asymptotic values
T=200;
T_ego=600;

legend

while true
data_Arduino=fscanf(a,'%s');
    data_split=strsplit(data_Arduino,'*');
    engine_state(j)=str2double(data_split(1));
    voltage(j)=str2double(data_split(2));
    state(j)=str2double(data_split(3));
    Tinj_OL(j)=str2double(data_split(4));
    Tinj_OP_A(j)=str2double(data_split(5));
    sum(j)=str2double(data_split(6));
    Cl(j)=str2double(data_split(7));
    Tinj_CL(j)=str2double(data_split(8))/(2^10);


%Sensors when engine is ON
```

```
if engine_state(j)==1
    if engine_state(j-1)==0
        temperature(j-1)=temperature_last_OFF;
        temperature_ego(j-1)=temperature_ego_last_OFF;
        speed(j-1)=speed_last_OFF;
        air_flow_rate(j-1)=MAF_last_OFF;
        T_error(j-1)=T_error_last_ON;

    end

    temperature(j)=(1-0.1)*temperature(j-1)+0.1*T;
    temperature_ego(j)=(1-0.1)*temperature_ego(j-1)+0.1*T_ego;
    potentiometer(j)=voltage(j)*100;
    air_flow_rate(j)=(1-0.1)*air_flow_rate(j-1)+0.1*(speed(j-1));



    if state(j)<4
        speed(j)=(1-0.4)*speed(j-1)+0.4*800;
    else
        speed(j)=(1-0.4)*speed(j-1)+0.4*potentiometer(j);
        if speed(j)<=700
            speed(j)=700;
        end
    end


%Error function and ptimal injection time
min=-200;
max=200;
r(j)=(max-min).*rand(1)+min;
T_error(j)=(1-0.1)*T_error(j-1)+0.1*r(j);
Tinj_OPT(j)=Tinj_OL(j)+T_error(j);

%Storing T_error last value when engine is ON
T_error_last_ON=T_error(j);


%EGO sensor virtualization
  if Tinj_CL(j)<=Tinj_OPT(j)
    EGO(j)=-1;
else
    EGO(j)=1;
end

%Conversion of sensors data to integers
temperature_int(j)=round(temperature(j)*2^10);
temperature_ego_int(j)=round(temperature_ego(j)*2^10);
speed_int(j)=round(speed(j)*2^10);
air_flow_rate_int(j)=round(air_flow_rate(j)*2^10);
Tinj_OPT_int(j)=round(Tinj_OPT(j));

%Writing data on serial

  data_Matlab=[temperature_int.' temperature_ego_int.' speed_int.'
  air_flow_rate_int.' Tinj_OPT_int.' EGO.'];
    str=sprintf("%d*%d*%d*%d*%d*%d*%d\n", data_Matlab(j,:));

    fprintf(a,'%s\n', str);
```

```
    %Engine states display

        switch state(j)
            case 1
                string_CR=sprintf("Iteration number %d: CRANKING", j-1);
                disp(string_CR);
            case 2
                string_WU=sprintf("Iteration number %d: WARM UP", j-1);
                disp(string_WU);
            case 3
                string_OL=sprintf("Iteration number %d: OPEN LOOP", j-1);
                disp(string_OL);
            case 4
                string_CL=sprintf("Iteration number %d: CLOSED LOOP", j-1);
                disp(string_CL);
            case 5
                string_HA=sprintf("Iteration number %d: HARD ACCELERATION", j-1);
                disp(string_HA);
            case 6
                string_D=sprintf("Iteration number %d: DECELERATION", j-1);
                disp(string_D);
            case 7
                string_I=sprintf("Iteration number %d: IDLE", j-1);
                disp(string_I);
            otherwise
                break
        end


        j=j+1;

      %Sensors and injection times when engine is OFF
    else if engine_state(j)==0
        disp("ENGINE OFF");
        temperature(j)=(1-0.1)*temperature(j-1)+0.1*25;
        temperature_ego(j)=(1-0.1)*temperature_ego(j-1)+0.1*25;
        speed(j)=(1-0.4)*speed(j-1);
        potentiometer(j)=0;
        air_flow_rate(j)=(1-0.4)*air_flow_rate(j-1);
        Tinj_CL(j)=0;
        Tinj_OL(j)=0;
        Tinj_OPT(j)=0;

      %Storing last values when engine is turned OFF
        temperature_last_OFF=temperature(j);
        temperature_ego_last_OFF=temperature_ego(j);
        speed_last_OFF=speed(j);
        MAF_last_OFF=air_flow_rate(j);


        j=j+1;

    end
    end

%Data plotting
figure (1);
subplot(3,2,1);
```

```
plot(temperature(:), 'r','DisplayName','Coolant Medium Temperature');
title('Coolant medium temperature');
xlabel('Iterations');
ylabel('°C');
hold on;
subplot(3,2,2);
plot(temperature_ego(:),'b','DisplayName','EGO Sensor Temperature');
title('Ego sensor temperature');
xlabel('Iterations');
ylabel('°C');
subplot(3,2,3);
plot(speed(:),'g','DisplayName','RPM sensor');
title('Crankshaft speed');
xlabel('Iterations');
ylabel('RPM');
subplot(3,2,4);
plot(air_flow_rate(:),'m','DisplayName','MAF sensor');
title('Air flow rate');
ylabel('mg/s');
xlabel('Iterations');
subplot(3,2,[5,6]);
plot(potentiometer(:),'k','DisplayName','Potentiometer');
title('Potentiometer voltage');
xlabel('Iterations');
ylabel('Volts');

legend('AutoUpdate','off')


figure(2);
plot(Tinj_OL(:), 'r','DisplayName','Tinj OPEN LOOP');
title('OPEN-LOOP, OPTIMAL AND CLOSED-LOOP INJECTION TIMES');
ylabel('ms');
xlabel('Iterations')
hold on;
plot(Tinj_OPT(:), 'b','DisplayName','Tinj OPTIMAL');
plot(Tinj_CL(:), 'g','DisplayName','Tinj OVERALL');

legend('AutoUpdate','off')

end
```

5.4.2 Arduino code

```
//Legenda

 //CRANKING state=1;
 //WARM UP state=2;
 //OPEN LOOP state=3;
 //CLOSED LOOP state=4;
 //HARD ACCELERATION state=5;
 //DECELERATION state=6;
 //IDLE state=7;

#include <elapsedMillis.h>
```

```
//Variables declaration

elapsedMillis timer;
const int switchPin=3;
const int potPin=A0;

int engine_state=0;
int state=1;
int future_state;
int voltage=0;
int j=2;
String temperature, temperature_ego,rpm, Tinj_OPT, air_flow_rate, EGO;
long temperature_num,temperature_ego_num,rpm_num,air_flow_rate_num, Tinj_OPT_num;
signed int EGO_num;


long Tinj;
long Tinj_OL;
long Tinj_CL;
long sum=0;
long Cl;
long k=-10; // 8/1024

//char data[100];

//Initialization engine variables


float stoichiometric_ratio=14.7; //Gasoline engine
float Rf=4.3; //g/s Gasoline delivery rate @ 3 bars
int constant=63;

void setup(){
  Serial.begin(115200);
  pinMode(switchPin,INPUT);
}

void loop(){

  if(timer>=1000){

  engine_state=digitalRead(switchPin);

  //If engine is ON, read the voltage and implement
  //the control algorithm, by reading data from Matlab
  //and by returning "state";


  if(engine_state==1){

    float potVal=analogRead(potPin);
    voltage=(potVal/1023.0)*5.0*10;

    //Read Matlab data and convert into numeric values
    if(Serial.available()){

      temperature=Serial.readStringUntil('*');
      temperature_ego=Serial.readStringUntil('*');
      rpm=Serial.readStringUntil('*');
```

```
      air_flow_rate=Serial.readStringUntil('*');
      Tinj_OPT=Serial.readStringUntil('*');
      EGO=Serial.readStringUntil('\n');

      temperature_num=temperature.toInt();
      temperature_ego_num=temperature_ego.toInt();
      rpm_num[j-1]=rpm.toInt();
      air_flow_rate_num[j-1]=air_flow_rate.toInt(); //moltiplicare
      Tinj_OPT_num[j-1]=Tinj_OPT.toInt();
       EGO_num=EGO.toInt();
    }

//Open loop injection time calculation
     if (rpm_num[j-1]>0){

       Tinj_OL[j]=pow(2,10)*60*air_flow_rate_num[j-1]/(constant*2*rpm_num[j-1]);

     }

     else{
      Tinj_OL[j]=0;
     }

//Closed loop injection time calculation
       sum=sum+EGO_num;

       Cl=k*sum;

       Tinj_CL=round(Tinj_OL*(1*pow(2,10)+Cl));

 //State machine
    switch (state){
      case 0:
        if (engine_state==0){
          future_state=0;
        }
        else{
          future_state=1;
        }
        break;
      case 1:
        if(rpm_num<750*pow(2,10)){
          future_state=1;
        }
        else{
          future_state=2;
        }
        break;
      case 2:
        if(temperature_num<105*pow(2,10)){
          future_state=2;
        }
        else{
          future_state=3;
        }
        break;
      case 3:
        if(temperature_ego_num<340*pow(2,10)){
          future_state=3;
```

```
      }
      else{
        future_state=4;
      }
      break;
    case 4:
      if(voltage<15){
        future_state=6;
      }
      else if(voltage>20){
        future_state=5;
      }
      else{
        future_state=4;
      }
      break;
    case 5:
      if(voltage>15 && voltage<20){
        future_state=4;
      }
      else if (voltage<15){
        future_state=6;
      }
      else{
        future_state=5;
      }
      break;
    case 6:
      if(voltage>15 && voltage<20){
        future_state=4;
      }
      else if(voltage>20){
        future_state=5;
      }
      else{
        future_state=6;
        if(rpm_num<750*pow(2,10)){
          future_state=7;
        }
      }
      break;
      case 7:
      if(rpm_num>750*pow(2,10)){
        future_state=4;
      }
      else{
        future_state=7;
      }
      break;
    default:
    break;
  }
  state=future_state;
}
else{
  voltage=0;
  state=0;
  Tinj=0;
}
```

```
    timer=timer-1000;

//Writing data on serial
Serial.print(engine_state);
Serial.print('*');
Serial.print(voltage);
Serial.print('*');
Serial.print(state);
Serial.print('*');
Serial.print(Tinj_OL[j]);
Serial.print('*');
Serial.print(Tinj_OPT_num[j-1]);
Serial.print('*');
Serial.print(EGO[j]);
Serial.print('*');
Serial.print(sum[j]);
Serial.print('*');
Serial.print(Cl[j]);
Serial.print('*');
Serial.print(Tinj[j]);
Serial.print('\n');

    j=j+1;

  }
 }
```

6. Conclusions

This project started from the analysis of different realities in terms of didactics all around the world: best quality universities providing Automotive Engineering courses have been taken into consideration and permitted to highlight the general lack in terms of first-hand experience for what concerns electronics teaching. In some cases, it has been pointed out that electronics is not even a mandatory subject foreseen by the courses, which usually prefer to give students a wide knowledge in terms of electric systems only for power-train design purposes with a view to decarbonization. On the other hand, this approach does not result to be in line with market demand: engine pollutant emissions control, traction and stability control, connectivity and many other functions of capital importance are all implemented by ECUs whose functioning is worth to be studied in depth in the context of Automotive Engineering courses, which should focus more on providing future engineers a wide perspective over microprocessors and basic programming languages, apart from mechanics and machine design: this could be extremely important also to prepare students to face properly the world of work, which is increasingly demanding in this sense.

The goal of this project was that of constituting a small step towards this auspicial change: the hope is that, by means of the designed prototype, Politecnico di Torino students may appreciate, during the Electronic Systems of Vehicle course, a view of the simplified functioning of a real ECU in the optics of emissions control of an internal combustion engine, whose programmed banning seems nowadays extremely premature.

The design flow of the prototype started from the understanding of the electronic architecture of the vehicle: necessary sensors, their position and their working principles have been described and analysed in order to simulate as realistically as possible how they are used by the ECU to assess the correct amount of fuel which should be injected inside the cylinders in each engine mode, according to power demand and proper conversion efficiency. From this point, then, the control algorithm was depicted and implemented on Arduino by means of the simulation of a state machine, where every state was set by imposing realistic thresholds for crankshaft rpm, coolant medium temperature, EGO sensor temperature and TPS voltage to shift from one state to the other. Control techniques, open-loop and closed-loop ones, have been explained and implemented as well, by means of MAF sensor and EGO sensor simulation.

To guarantee a proper functioning of the prototype, a serial communication has been established,

between MATLAB, where sensors have been simulated and plots have been generated, and Arduino, where sensors data have been stored and the state machine implemented, also by taking into consideration the voltages of TPS and ON/OFF slide switch, assembled on Arduino board. Serial communication principles have been discussed, with a main focus on UART communication protocol, the one actually adopted to establish a data exchange between MATLAB and Arduino, by means of an USB cable. Serial communication principles have been discussed, with a main focus on UART communication protocol, the one actually adopted to establish a data exchange between MATLAB and Arduino, by means of an USB cable. In this sense, a more sophisticated and realistic system may be created in the future by relying on a CAN bus, in order to replicate the solution which is actually adopted in vehicles. Another important simplification which has been adopted in design the prototype that could be improved in the future may be related to the functions used to simulate sensors behaviours: even if in some cases, as for temperature sensors, the transient function behaviour may be pretty realistic, for functions as that of the MAF and the speed, the simulated approach results to be extremely not consistent with the reality. This is mainly due to the fact that usually vehicles are equipped with the gearbox, which provides different gear ratios which affects crankshaft speed according to engine load and desired torque to be delivered to wheels. In this case, for sake of simplicity, a single gear ratio vehicle has been considered, so that to set a single dependency of the speed function on the TPS voltage and, in turn, of the MAF sensor on the speed function. A good approach to overcome this issue could be that of importing inside the model a homologation cycle to represent speed and gears changes, as WLTP cycle (i.e., Worldwide Harmonized Light Vehicles) or NEDC (i.e., New European Driving Cycle).

Net of these simplifications, the model could represent a first approach for students to the logic of an ECU and could be considered a useful didactical tool also in the view of potential improvements and changes which can be implemented on the prototype.

7. Acknowledgements

Desidero ringraziare il Professor Paolo Crovetti per avermi concesso questo lavoro di tesi, per essere stato un insegnante dall'insostituibile chiarezza e disponibilità e per avermi seguita in questo lavoro passo dopo passo, con pazienza e precisione.

Ho scelto di dedicare questa tesi a mia nonna Amorina, che ha sempre costituito il faro a cui guardare nei momenti di sconforto e la ragione per andare avanti in questo percorso nonostante le difficoltà; difficoltà che – prima di me – ha visto affrontare con successo da mio padre, che ho sempre considerato il mio punto di riferimento e la profonda ragione della mia scelta di intraprendere questo percorso universitario. Grazie Papà, per aver creduto che ce l'avrei potuta fare, per aver sempre vegliato su di me col tuo sguardo dolce e premuroso e per essere stato sempre una presenza silenziosa ma forte al mio fianco. Grazie Mamma, per non essere al contrario mai stata una presenza silenziosa, ma per aver invaso la mia vita di rumore e di gioia e per avermi cresciuta col desiderio di fare qualcosa di importante, così da essere sempre il più possibile aderente alla visione che ho della persona che desidero essere. Grazie Chicco, per essere stato nominalmente un fratello minore, ma nella realtà dei fatti un esempio di tenacia e coraggio che mi ha ispirata nel tempo e che mi ha fatto capire l'importanza di credere in qualcosa, nelle mie capacità, nella possibilità di un futuro gioioso e soprattutto nella certezza di poter contare sempre su di te, come tu potrai sempre su di me, nonostante un mare ci separi.
Grazie Federico, per aver cambiato il volto alla mia esistenza e per essere stato un compagno meraviglioso nella vita e in quest'ultimo tratto di studi che ho avuto l'immensa fortuna di poter condividere con te, sapendo di poter contare sul tuo aiuto e sul tuo supporto, effettivo e morale.
Grazie Santuzza, Andrea e Roberto per avermi accolto nelle vostre vite come una figlia e per avermi saputo dare un amore infinito che ricambierò sempre e di cui cercherò di essere sempre all'altezza: anche il vostro contributo è stato fondamentale per il conseguimento di questo titolo; siete stati un'ancora e una casa.
Spero di avervi resi tutti orgogliosi in qualche modo e di essere riuscita a ripagare i tanti sforzi e le tante fatiche che tutti, in un modo o nell'altro, avete deciso di dedicarmi.

8. References

1 https://edurank.org/engineering/automotive/

2 https://www.tsinghua.edu.cn/en/

3 https://umich.edu/

4 https://www.wisc.edu/

5 https://en.sjtu.edu.cn/

6 https://www.mit.edu/

7 https://english.bit.edu.cn/

8 https://www.osu.edu/

9 https://www.en.aau.dk/

10 https://www.berkeley.edu/

11 https://en.tongji.edu.cn/

12 https://global.jlu.edu.cn/

13 https://www.rwth-aachen.de/go/id/a/?lidx=1

14 http://www.tju.edu.cn/english/index.htm

15 https://en.swjtu.edu.cn/

16 https://www.tamu.edu/

17 https://www.chalmers.se/en/Pages/default.aspx

18 https://www.tudelft.nl/

19 https://www.u-tokyo.ac.jp/en/

20 https://vt.edu/

21 Electric and Electronic Systems for Vehicles, course slides 2021-2022, Paolo Crovetti

22 Internal Combustion Engine Fundamentals, 2nd edition, 2018, John B. Heywood

23 Internal Combustion Engine Fundamentals, 2nd edition, 2018, John B. Heywood

24 Internal Combustion Engine Fundamentals, 2nd edition, 2018, John B. Heywood

25 Electric and Electronic Systems for Vehicles, course slides 2021-2022, Paolo Crovetti

26 Open Loop System, 2021, Ravi Teja

27 Closed Loop System, 2021, Ravi Teja

28 Electric and Electronic Systems for Vehicles, course slides 2021-2022, Paolo Crovetti

29 https://ipcsautomation.com/encoder-working-principle-theory-explanation/

30 Advanced Vehicle Technology, 2nd edition, 2002, Heinz Heisler MSc., BSc., F.I.M.I., M.S.O.E., M.I.R.T.E., M.C.I.T., M.I.L.T.

31 Electric and Electronic Systems for Vehicles, course slides 2021-2022, Paolo Crovetti

32 Process Zirconia Oxygen Analyzer — State of Art Zirkondioxid-Sauerstoffsensoren — Stand der Technik, 2010, Pavel Shuk

33 Electric and Electronic Systems for Vehicles, course slides 2021-2022, Paolo Crovetti

34 Arduino® UNO R3 Datasheet, 2022

35 ATmega328P Datasheet, 2015, Atmel Corporation

36 Arduino® UNO R3 Datasheet, 2022

37 Arduino UNO Potentiometers Datasheet, ACP Technologies

38 Arduino UNO Potentiometers Datasheet, ACP Technologies

39 Arduino UNO Potentiometers Datasheet, ACP Technologies

40 Arduino UNO Potentiometers Datasheet, ACP Technologies

41 https://moniteurdevices.com/knowledgebase/knowledgebase/what-is-the-difference-between-spst-spdt-and-dpdt/

42 https://moniteurdevices.com/knowledgebase/knowledgebase/what-is-the-difference-between-spst-spdt-and-dpdt/

43 https://moniteurdevices.com/knowledgebase/knowledgebase/what-is-the-difference-between-spst-spdt-and-dpdt/

44 https://moniteurdevices.com/knowledgebase/knowledgebase/what-is-the-difference-between-spst-spdt-and-dpdt/

45 UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter, Eric Pena and Mary Grace Legaspi

46 UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter, Eric Pena and Mary Grace Legaspi

47 UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter, Eric Pena and Mary Grace Legaspi

48 UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter, Eric Pena and Mary Grace Legaspi

49 Combustion-Related Emissions in SI Engines, 2021, Zhao Youcai, Wei Ran

50 Investigation of Auto-Ignition of Several Single Fuels, 2014, A.A.R. Aziz