Bachelor’s Final Degree Project

Development of a Motor Controller-based System for Data Monitoring and Real-time Display of Electric Formula Student Car

ARTURO JULVE BLANCO

Bachelor’s Degree in Automotive Engineering

Supervisor: Dr. Moises Garín Escrivá

Granollers | June 2023

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to the people involved in some kind of way in this dissertation.

In first place, thank Moises Garín, the supervisor of the project and UTRON's team advisor, for the technical support and guidance through all the stages of the process. At the same time, I would like to acknowledge Jesús Moreno's work and the University of Vic for making sure all the components I needed were ready at the right moment. I cannot forget about the hard work Juana Alcalde, the janitor, consistently put on helping me when I needed and keeping my work environment at the workshop in good shape.

I would like to thank UTRON Racing Team members as well for their commitment to this shared passion and for making long workdays enjoyable. With a special mention to former teammates and mentors, Martí Galbany and Oriol García, and current members that worked directly in some areas of the project, Biel Aren, Nil Brugada, David López and Carlos Ortiz.

Unconditionally thankful to my family for always being by my side no matter the situation and with a crazy desire to make me the happiest. Out of all of them I would like to remark on my sister's participation and advice. Thank you, Andrea, for taking over the esthetic part of the dissertation.

Finally, I would like to thank all the friends, colleagues, and other individuals who provided encouragement and support throughout the project. Your belief in my vision and your inspirational words kept me motivated during challenging times.

Thank you all for your invaluable contributions in making this project a success.

ABSTRACT

Title: *Development of a Motor Controller-based System for Data Monitoring and Real-time Display of Electric Formula Student Car*

Author: Arturo Julve Blanco

Supervisor: Dr. Moises Garín Escrivá (UVic)

Date: June 2023

Keywords: Electric Formula Student Car, Motor Controller, Python, Data Logging, Graphical User Interface (GUI), Dashboard, Performance Optimization

This dissertation is part of an electric Formula Student prototype development with the objective of creating a system that tracks the behavior of the car in real-time to allow for analysis and improvement.

The project focuses on the motor controller component of the powertrain system and is divided into three main parts: preparation of powertrain elements, development of a data logging program, and real-time display of data on a dashboard with a Graphical User Interface.

The developed system proved to be effective in tracking the car's behavior and providing real-time feedback to the driver and the team, leading to improved performance and efficiency.

Video link: https://canal.uvic.cat/Player/B66eC3a5

RESUM

Títol: *Desenvolupament d'un Sistema Basat en la Controladora del Motor per la Monitorització i Visualització a Temps Real de Dades d'un Cotxe Elèctric de Formula Student*

Autor: Arturo Julve Blanco

Tutor: Dr. Moises Garín Escrivá (UVic)

Data: Juny de 2023

Paraules clau: Cotxe de Formula Student Elèctric, Controladora de Motor**,** Python, Registre de Dades, Interfície Gràfica (GUI), Dashboard, Optimització de Rendiment

Aquest treball de fi de grau forma part del desenvolupament d'un prototip elèctric de Formula Student i té com a objectiu crear un sistema que registri el comportament del cotxe en temps real per permetre l'anàlisi i la millora.

El projecte es centra en la controladora del motor en el sistema de tren de potència i es divideix en tres parts principals: la preparació dels elements del tren de potència, el desenvolupament d'un programa de registre de dades i la seva visualització en temps real en un quadre de comandament amb una interfície gràfica.

El sistema desenvolupat ha demostrat ser eficaç en el seguiment del comportament del cotxe i en proporcionar informació en temps real al conductor i a l'equip, el que ha portat a una millora del rendiment i l'eficiència.

Enllaç del video: https://canal.uvic.cat/Player/B66eC3a5

GLOSSARY

1. **FS:** abbreviation commonly used to refer to Formula Student, an international engineering competition for university students.
2. **HORIZON 20:** name that the first prototype of UTRON Racing receives after the year its development started.
3. **Automotive Talent Show:** event that promotes young talent inside the automotive industry. It takes place in the Circuit of Barcelona-Catalunya once a year.
4. **CAD:** stands for Computer-Aided Design and it is a technology used in various industries, including engineering, architecture, and product design, to create precise and detailed digital representations of physical objects or systems.
5. **EMRAX:** brand of electric motors commonly used in the field of electric vehicles (EVs) and hybrid electric vehicles (HEVs). The EMRAX motors are designed and manufactured by a Slovenian company.
6. **CASCADIA:** American company that develops and manufactures inverters, traction motors, and geartrains used in a variety of electric vehicle applications ranging from Formula One to heavy duty on-highway truck.
7. **MOSFET:** short for Metal-Oxide-Semiconductor Field-Effect Transistor, is a type of semiconductor device commonly used in electronic circuits. It operates as a voltage-controlled switch, allowing or blocking the flow of current based on the voltage applied to its gate terminal.
8. **ECU:** stands for Engine Control Unit. Term to refer to an electronic control module or computer system that manages and controls various aspects of an engine's operation.
9. **QEV:** Catalan company pioneer in the field of electric mobility and electric motorsport series.
10. **RPM:** stands for Revolutions Per Minute. It is a unit of measurement used to quantify the rotational speed of an object, typically an engine or a rotating component.
11. **Fuse:** protective device used in electrical circuits to prevent damage from excessive current flow. It is a small component that contains a metal wire or strip that melts when subjected to a current higher than its rated value.
12. **EMI:** refers to the disturbance caused by electromagnetic radiation from one electronic or electrical device that affects the operation of another device.
13. **CAN:** stands for Controller Area Network. It is a communication protocol widely used in automotive and industrial applications for data exchange between electronic devices or modules within a network.
14. **AMPSEAL:** type of electrical connector system manufactured by Tyco Electronics Connectivity.
15. **DB-9:** common type of connector used for serial communication. The "DB" stands for "D-subminiature," and the number "9" indicates the number of pins or contacts in the connector.

16. **Wedge Lock:** mechanical fastener that provides a strong and reliable connection by expanding and gripping the surrounding objects.
17. **RMS:** abbreviation referred to the company Rinehart Motion Systems. It is a subsidiary of Cascadia Motion.
18. **EEPROM:** stands for Electrically Erasable Programmable Read-Only Memory. It is a type of non-volatile memory that can be electrically erased and reprogrammed.
19. **ASCII:** American Standard Code for Information Interchange. It is a character encoding standard used to represent text and symbols in digital form.
20. **CSV:** stands for Comma-Separated Values. It is a plain-text file format commonly used for storing and exchanging tabular data. In a CSV file, each line represents a row of data, and the values within each row are separated by commas.
21. **LCD display:** stands for Liquid Crystal Display. It is a flat panel display technology that uses liquid crystals to create images or text on a screen.
22. **Linux:** free and open-source operating system kernel that serves as the foundation for various Linux-based operating distributions.
23. **HDMI:** stands for High-Definition Multimedia Interface. It is a digital interface commonly used for transmitting high-quality audio and video signals between devices. HDMI cables and ports are found on various consumer electronics.
24. **ELECROW:** Chinese company that specializes in the design, manufacturing, and distribution of electronic components, development boards, and custom electronic products. They offer a range of services and products to support makers, hobbyists, and professionals in the field of electronics.
25. **GIMP:** stands for GNU Image Manipulation Program. It is a free and open-source raster graphics editor that allows users to edit and manipulate digital images. GIMP offers a wide range of tools and features for tasks such as photo retouching, image composition, and graphic design.
26. **API:** set of rules and protocols that allows different software applications to communicate and interact with each other. (Application Programming Interface)
27. **RGB:** stands for Red, Green, and Blue. It is a color model commonly used in digital imaging and display systems to represent and display a wide range of colors. In the RGB model, colors are created by combining different intensities of red, green, and blue light.
28. **Idle:** in computer systems, "idle" refers to the state of a system or a process when it is not performing any significant tasks or computations. The idle state often occurs when a computer or a program is waiting for user input or when it has completed all the assigned tasks and is awaiting further instructions.

TABLE OF CONTENTS

ACKNOWLEDGMENTS 1
ABSTRACT 2
RESUM 3
GLOSSARY 4
TABLE OF CONTENTS 6
LIST OF FIGURES 8
1. INTRODUCTION 9
1.1. FORMULA SAE 9
1.1.1. Overview 9
1.1.2. FS Rules 10
1.1.3. The Competition 11
1.2. UTRON RACING 13
1.2.1. History 13
1.2.2. Structure 14
1.2.3. Car features 15
1.3. HORIZON 20 POWERTRAIN 16
1.3.1. Design requirements 16
1.3.2. Engine 17
1.3.3. Controller 17
1.3.4. Battery pack 18
1.4. PROJECT STATUS 20
2. OBJECTIVES 21
3. STATE OF THE ART 22
3.1. ELECTRIC POWERTRAIN TECHNOLOGY 22
3.2. DASHBOARD TECHNOLOGY 23
4. PROJECT DEVELOPMENT 25
4.1. CONTROLLER RESEARCH 25
4.1.1. Controller connections 27
4.1.2. Material purchasing 28
4.1.3. Assembly 29
4.1.4. Controller software 30
4.1.5. Data Acquisition Program 31
4.2. DATA LOGGING 32
4.2.1. Data Selection 33
4.2.2. CSV and Openpyxl Libraries 33
4.2.3. Code explanation 34

4.3. DASHBOARD 35
4.3.1. Dashboard interface design 35
4.3.2. Dashboard hardware 37
4.3.3. GTK library 39
4.3.4. Dashboard code explanation 40
4.4. CODE INTEGRATION 42
4.4.1. Threading 42
4.4.2. Code Adaptation 42
4.5. TESTING 44
5. CONCLUSION 46
5.1. RESULTS 46
5.2. FUTURE STEPS 47
6. BIBLIOGRAPHY 49
7. ANNEXES 51
ANNEX A 51
Project planning 51
ANNEX B 52
AMPSEAL J1-35p connections 52
AMPSEAL J2-23p connections 53
ANNEX C 54
Serial Comunication Code 54
ANNEX D 56
Data Acquisition Parameters 56
ANNEX E 57
Motor speed to vehicle speed conversion 57
ANNEX F 58
Data Logging Code 58
ANNEX G 64
Display housing general dimensions 64
ANNEX H 65
Graphical User Interface Code 65
ANNEX I 79
UTRON_Excel widget 79
ANNEX J 83
UTRON_Logger widget 83
ANNEX K 88
UTRON_Dashboard APP 88

LIST OF FIGURES

Figure 1. Formula Student Teams at Formula Student Germany…..………………………….….....9

Figure 2. Scrutineering at Formula Student Spain Competition………………………………....10

Figure 3. Acceleration Event at Formula Student Spain Competition………………..................12

Figure 4. UTRON Racing Members at the Automotive Talent Show…..………..…..................13

Figure 5. HORIZON 20 CAD Design……………….……………………….....……………....15

Figure 6. HORIZON 20 Powertrain Components in CAD Design…………………..................16

Figure 7. EMRAX 208 MV Engine……………………………..………………….…………....17

Figure 8. Motor Controller Basic Circuitry……………………...……..…...…………………....18

Figure 9. LG H2 Li-ion Battery Modules…...………………...……………………………….....19

Figure 10. HORIZON 20 Manufacturing Process………………….………...…….…………....20

Figure 11. Formula E GEN 3 Car Design…...…………………………...…………………….....23

Figure 12. Formula 1 Team Racing Simulator…………………………………...……………...24

Figure 13. PM100 Typical Wirings………………………………...…………….……………....25

Figure 14. Controller's Pre-Charge Circuit Morphology…...……………………..……………26

Figure 15. EMRAX Motor Connections…...………………………...…………………….…....27

Figure 16. 35 pins AMPSEAL Connector…...……………………………….…………………....28

Figure 17. 23 pins AMPSEAL Connector…...……………………………….…………………....28

Figure 18. AMPSEAL Connectors Connection Procedure Scheme…...…………...…………....29

Figure 19. HORIZON 20 Powertrain Model………………….…………….....………………....30

Figure 20. Controller Software Main Window…...…………………………...……………….....31

Figure 21. Controller's Serial Port Settings……………………………..…….…….…………....32

Figure 22. Formula 1 Dashboard………………………………..………………….…………....35

Figure 23. UTRON's Dashboard Layout………………….………………….........................36

Figure 24. Display Housing CAD Design……………….……………...……………………....38

Figure 25. Dashboard Panel General Dimensions………………….………………...………....38

Figure 26. Dashboard Final Assembly……………….…...……………...…...………………....39

Figure 27. Final Code Diagram…………………….………...….………...……………….....44

Figure 28. Progressive Acceleration Graph……………………………………..……………45

Figure 29. Hard Braking Graph……………….…………………………………....………….....45

Figure 30. Dashboard Final Result……………….………………...……………………………....47

1. INTRODUCTION

This first chapter is meant to expose key concepts that surround the project and at the same time are the basis to fully understand the meaning and the development of it. They are terms that are not part of pure development but influence it in many ways. In other words, the Formula Student competition, the team, previous designs, and the HORIZON 20[2] status itself have defined the frontiers of the work that has been carried out.

1.1. FORMULA SAE

UTRON Racing is University of Vic's bet to enter in Formula SAE. It is an international engineering competition in which teams are challenged to design, finance, manufacture and drive a single-seater car. The main goal is to race the prototype developed against other teams in a series of events. (In Figure 1 the multitude in Formula Student events can be seen)

FIGURE 1. FORMULA STUDENT TEAMS AT FORMULA STUDENT GERMANY. (IMAGE FROM [2])

1.1.1. Overview

The competition was founded in 1979 by the Society of Automotive Engineering (SAE) in the United States and it has been popular from the beginning. In 1998, the first competition on European soil was held, more precisely in the United Kingdom. But it was not until 2010 that the Society of Automotive Technicians (STA), together with important automotive companies, organized the first edition of Formula Student Spain (FSS). Currently, there are 20 FS[1] competitions around the world, with more than 800 teams. The Spanish tournament attracts more than 100 teams from 19 different countries.

These teams not only have to create a vehicle to compete, but they also must accomplish a lot of self-management work, as they are additionally responsible for the marketing, the acquiring of sponsors and the financial planning. Those together with competitiveness, are new automotive challenges that students must deal with to ultimately succeed.

Lately, FS also promotes forward thinking and future technologies. In 2010, Formula Student Combustion (FSC) was complemented by Formula Student Electric (FSE), and in 2017 Formula Student Driverless (FSD) was added. The current trend in racing is towards electrification and the implementation of autonomous driving systems, thus reducing internal combustion.

1.1.2. FS Rules

As the bigger motorsport competitions, Formula Student is ruled by a series of points and requirements [1]. Since it started the rules have been very clear, but every year they are updated to avoid misunderstandings, correct errors, and add new restrictions. Once they are launched the job for the teams is to make sure they are strictly following them and avert any controversial situation.

The rules are established to ensure two basic things. In first place and the most important one is guaranteeing safety. Through the requirements of design and behavior, some limits are set to create a safe environment during the events. Not all the designs are allowed, this being evaluated by the scrutineers, and a good conduct must be satisfied at all costs. Meanwhile, in second place, the rules pursue the fairness during the competition. It is good to know that all teams are ruled by the same criteria and have the same chances to win when the event starts. If the rules are broken the consequences can go from a reduction of points in the total score to the disqualification of the event for years. (Figure 2 shows how the scrutineering prior to the events is performed)

FIGURE 2. SCRUTINEERING AT FORMULA STUDENT SPAIN COMPETITION. (IMAGE FROM [3])

Apart from that, the rules can be very helpful for rookie teams. They describe the basics inside the engineering branches that are legal so they can be found as a guideline to start developing the project.

The fundamentals of the rules are always the same but different versions are launched depending on the event. Focusing on UTRON Racing, the rules followed must be the ones from Spain, but they are only a small variation from the Formula Student Germany rules, which is the biggest competition in Europe.

1.1.3. The Competition

When in the competition [2][3], and knowing that the car is legal to compete, the teams take part of the static and dynamic events.

Static events are these where the vehicle does not compete on the track. It involves submitting documentation related to the vehicle and presenting it in front of the judges. The static events are:

- <u>Business Plan Presentation:</u> The objective of the BPP is to analyze the team's ability to develop and deliver a comprehensive business model which demonstrates their product (a prototype race car) could become a rewarding business opportunity that creates a monetary profit. The judges are treated as if they were potential investors or partners for the presented business model. It must relate to the specific prototype race car entered in the competition. The quality of the actual prototype is not considered as part of the BPP judging.
- <u>Cost and Manufacturing Event:</u> The cost and manufacturing event aims to evaluate the team's understanding of the manufacturing processes and costs associated with the construction of a prototype race car. This includes trade off decisions between content and cost, make or buy decisions and understanding the differences between prototype and mass production.
- <u>Engineering Design Event:</u> The concept of the design event is to classify the students' engineering processes and efforts that went into the design of a vehicle, meeting the intent of the competition. Proprietary components and systems that are incorporated into the vehicle design as finished items are not evaluated as a student designed unit but are only assessed on the team's selection and application of that unit.

Dynamic events are those in which the vehicle competes on the circuit. It means that the performance of the car and the skills of the driver speak by themselves. The dynamic events are:

- <u>Skidpad Event:</u> The objective of the Skidpad event is to measure the car's lateral grip on a flat surface while making a constant radius turn. The track consists of two pairs of concentric circles in a figure of eight pattern.
- <u>Acceleration Event:</u> The Acceleration event evaluates the car's acceleration in a straight line, from a standing start, over 75 m. This event is divided in two heats

that must be driven by two drivers, each of them has two attempts. The score is given by the difference between the best time and the worst one. (A shot of the Spanish Acceleration Event is captured in Figure 3)

FIGURE 3.ACCELERATION EVENT AT FORMULA STUDENT SPAIN COMPETITION. (IMAGE FROM [3])

- Autocross Event: Autocross tests the cars' dynamic ability in a one lap sprint. Two drivers are given two attempts at the course. The objective of the autocross event is to evaluate the car's maneuverability and handling qualities on a tight course without the hindrance of competing cars. The autocross event will combine the performance features of acceleration, braking, and cornering into one event.
- Endurance Event: The Endurance event is 22 km long driven on the same track as Autocross. Using two drivers, each of them drive half of the distance with a mandatory pit stop at the midpoint. The car must stop and start under its own power and no refueling or repairs are allowed. A team must finish Endurance to earn any points from Fuel Efficiency or Endurance. Consistency and reliability are key for this event.
- Efficiency Event: The car's fuel or power efficiency is measured during the Endurance event. Fuel or power usage and lap times are combined to determine how efficiently the car uses fuel. Here, a compromise between speed and power must be found.

The scoring of each of the events and all the details about the procedure are also defined in the rules. When all the trials are finished the judges have gathered enough information to proclaim a winner in each event and a winner overall.

1.2. UTRON RACING

In order to understand how the dissertation falls properly into the Formula Student world it is necessary to know more about UTRON Racing [4]. Its history, its resources, the personnel, the principles, and the design, among others, define the things that are developed later.

FIGURE 4. UTRON RACING MEMBERS AT THE AUTOMOTIVE TALENT^3 SHOW. (IMAGE FROM [4])

1.2.1. History

The team was founded in 2019 by students at the University of Vic that were enrolled in the Automotive Engineering Degree. Soon other disciplines joined to make a better and more complete team. This enabling the team to explore all the different fields required to start a project of this magnitude. (A picture of the current members of the team is shown in Figure 4)

From the very beginning the aspirations have been very clear. On top, there is the main goal of creating a car capable of succeeding in the previously described as dynamic events. But along the way, the team aims to autonomously learn and empower the passion for motorsport that is inside its integrands. Creating in that way a community inside the university that finds the perfect balance between learning about things that already exist and being able to see what is coming next in the future of technology.

1.2.2. Structure

After several methodologies and some generations that have come through the team, the size that is usually managed is between fifteen and twenty students. It is a reduced number of members compared to other experienced teams but very manageable for a team that is still in growth.

The current structure of the team is divided in six different departments, three of them dedicated to more technical aspects of the car and three focused on complementary disciplines but equally needed to achieve the final goals. In exception to other teams the roles are not as strict, the personnel can decide about other departments and move to where help is needed. The departments' name and their contribution to the team are as follows:

- Design and Manufacturing Department: It is the department responsible for the design of new parts and the physical assembly of the car, including other departments' components.
- Dynamic Systems Department: It is in charge of the bigger systems of the car like the suspension, the braking and the steering system. It is closely related to the design department, but it is focused on the preparation and performance of this systems rather than its design and fitting.
- Powertrain and Electronics Department: It involves all the electric parts of the car. By being an electrically powered vehicle, it means that both, the high voltage, and low voltage circuitry must be taken into account.
- Treasury Department: It oversees the accountancy of the project. It means keeping records and controlling what is received from the sponsors and how much the team can spend. In the competition this is what will be needed during the Cost Event.
- Media Department: Its main goal is to create a nice and trendy image of the team. It can refer to many different activities like social media activity and posts, arrangement of events, university shows, merchandising, interaction with sponsors…
- Business Plan Department: For the Business Plan Event an exclusive team is created. This group of members develops, during the whole year, what will be presented as an idea of business.

Apart from all the specific roles and leaders inside the departments, there is a higher role in the structure of the team. Leading the project and the members there is the figure of the Team Manager. He takes care of the people, the project and it's the maximum representation towards the sponsors, the competition and the university personalities. In my latest years that is the position I am taking, even though once I was a common member inside a department like the rest of teammates.

1.2.3. Car features

From the technical perspective a Formula SAE car can be defined in many ways. The design decision over the most important parts of the vehicle creates what is called a concept. Teams choose from many different specifications concerning their capabilities. Once teams have finished the manufacturing of a car they can follow two paths. In first place they can decide to continue with the same concept, meaning they will continue working with the same main components, or they can choose to switch to a different concept, usually with a greater range of improvement, but having to reinvent the whole car.

The combinations are countless but the parts that define the Horizon 20 are the following. Starting with the primary structure it is a tubular spaceframe made of steel. The bodywork is a glass fiber surface that covers the frontal part of the car. For this first concept no wings or aerodynamic devices of any type are considered.

Introducing the dynamic systems, the suspension goes directly attached to the suspension arms. This differs from the common concepts of pull rod and push rod that are predominant in motorsport. The drivetrain is simplified with a direct connection between the rear wheels and the rear gear, this meaning that no differential is implemented. Despite that, a reduction of the engine speed is made before connecting with the wheels. (CAD[4] overall design is displayed in Figure 5)

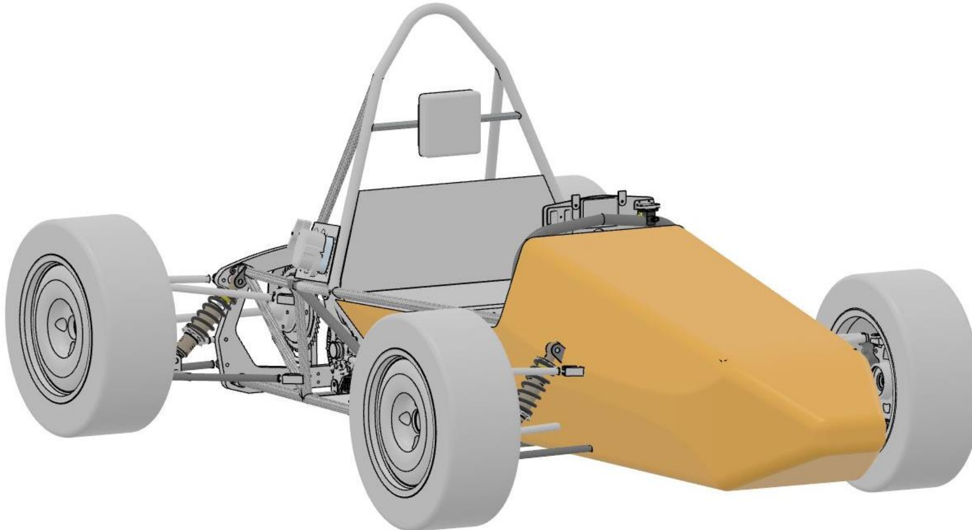


FIGURE 5. HORIZON 20 CAD DESIGN. (IMAGE FROM [OWN ELABORATION])

Finally, since it is an electric vehicle, the powertrain can also be defined by its morphology [5]. UTRON Racing chose an electrically powered car with a single engine which connects and only gives traction to the rear wheels. The control of the engine is done thanks to a controller, which will be described in detail during the project, and the power supply comes from a battery accumulator made of Lithium cells.

1.3. HORIZON 20 POWERTRAIN

The powertrain is the main system in which the dissertation is focused. It is convenient to describe what is in it and why it was designed that way [6] [7]. (Figure 6 describes the position of powertrain components inside the CAD assembly)

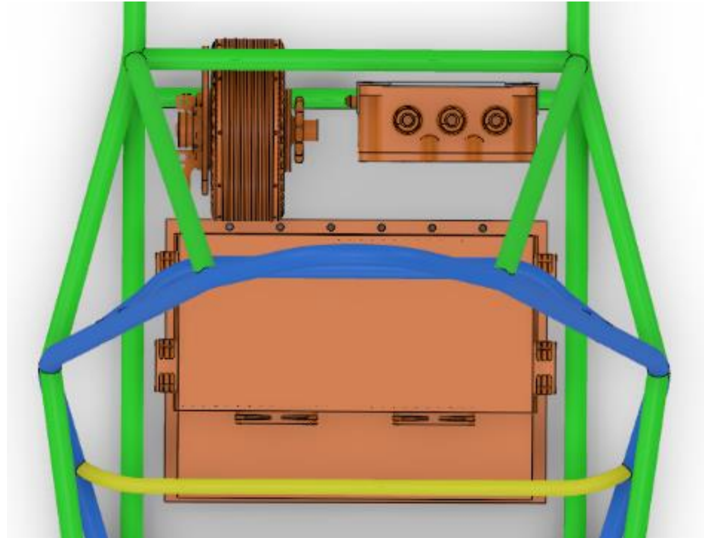


FIGURE 6. HORIZON 20 POWERTRAIN COMPONENTS IN CAD DESIGN. (IMAGE FROM [OWN ELABORATION])

When approaching the task of designing a powertrain from scratch the first thing to bear in mind is the final purpose. In UTRON's case as a formula student team, the primary goal is to power the car and bring it to the finish line in all the events of the competition. To achieve this, the system must meet some performance and safety requirements.

1.3.1. Design requirements

The event that pushes electric powertrains to the limit is the endurance race. Among all the events, it is the one that shows the true autonomy of cars. As described previously, the prototypes must be able to complete 22 laps around a track of approximately 1 kilometer long. It is no surprise that the entire design of the car is made thinking about this part. Other characteristics such as top speed or vehicle dynamics are improved later once the autonomy is fulfilled.

Moreover, an electric powertrain capable of giving motion to a vehicle of around 250 kilograms is a powerful system. Powerful when there is electricity involved means that there is danger as well and all caution could not be enough. For that reason the organizations ensure proper and safe designs through the rules that apply to all electric cars. This is an aspect that defines the powertrain you want to design as well.

These rules go through all the different components or parts that the electric system must have and set certain limits. It can go from the way things should be connected to the amount of energy components can store all the way through safe procedures when manipulating the system.

1.3.2. Engine

Even though it is the last component of the powertrain chain it is the one that defines the rest. The accumulator, the controller and even the transmission system depend on the characteristics of the electric engine. Because of its high continuous torque and high power/torque density, the engine selected by the team was the EMRAX[5] 208 MV [8]. (In Figure 7 the engine can be seen)

FIGURE 7. EMRAX 208 MV ENGINE. (IMAGE FROM [10])

It is a permanent magnet synchronous electric motor [9] model named after its diameter dimensions (208 mm). Mechanically speaking it is an axial flux type made of 10 pair of poles and the motor speed and position is measured with a one pole pair resolver. The resolver is an electromagnetic transducer that gives an electrical signal proportional to an angle depending on the AC signal sensed.

The most remarkable features and the ones that will define the rest of components are the following [10]:

- Its design nominal voltage which is 390 V.
- Its continuous current which is 220 A_{RMS} and it can reach peaks of 400 A_{RMS}.
- A peak power of 86 kW at 6000 RPM[10] and a continuous power up to 56 kW.
- A peak torque of 150 Nm and a continuous torque up to 90 Nm.
- Its efficiency that runs between 92% and 96%.

1.3.3. Controller

The CASCADIA[6] Motion PM100 DX is commonly known as an inverter [11][12]. The way it is designed allows the user to also control several parameters of the engine. This means it actually contains two main circuits inside: the three-phase inverter for the power side and a processor for the signals acquisition and the control side.

The PM100 DX transforms the direct current that comes from the accumulator into a three-phase altern current for the correct performance of the motor [13]. This is achieved mainly thanks to the combination of controlled switches (MOSFET[7] transistors) shown in Figure 8 and pulse width modulation also known as PWM technique.

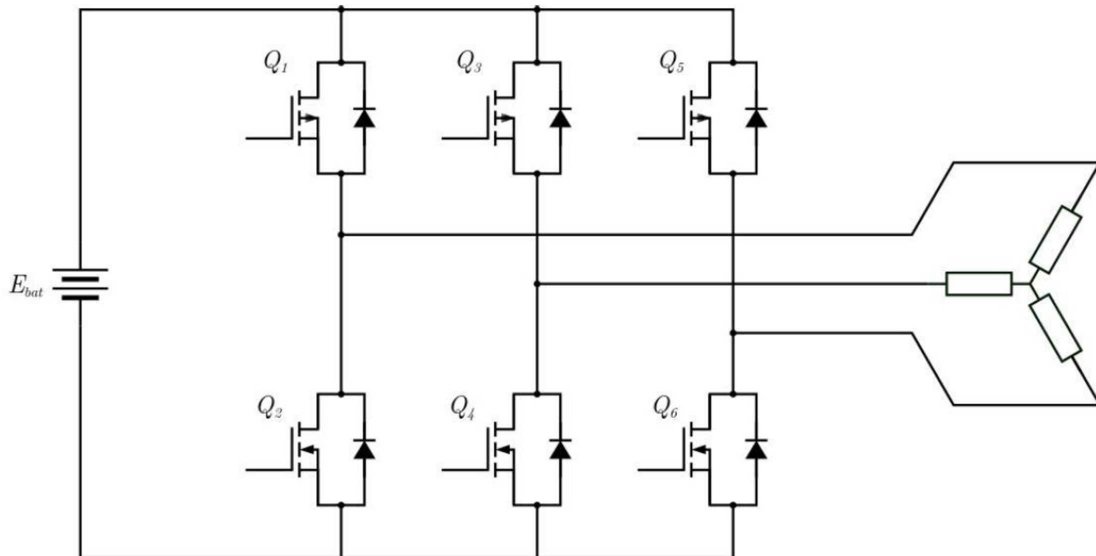


FIGURE 8. MOTOR CONTROLLER BASIC CIRCUITRY. (IMAGE FROM [13])

As for the ECU[8] side it can manage several inputs and outputs to monitor the state of the different components of the car apart from the battery and the engine. Software is also available to look into the configuration of the inverter and the motor and see real time data.

The features that best describe it are its operating DC voltage that goes from 50 to 400 VDC, the motor continuous current which is 300 A_{RMS} and its output power peak which is 120 kW.

1.3.4. Battery pack

As the final element of the powertrain we have the battery. It is the last part to be designed because it depends on the specifications of the vehicle, the race, the engine and the inverter. The company QEV[9] helped with the calculation of the requirements for the battery. Through their software they are able to guess the total energy the battery needs to store [14].

In the Horizon's powertrain the engine and the controller require an average voltage of 350 V. On the other side the calculation of the energy required showed that 7.3 kWh were needed. As a final calculation to obtain the total capacity we divide the amount of energy by the voltage. In our case the battery we want to obtain needs 20.85 Ah of capacity. From these three parameters the pack can start being designed.

Due to the dimensions of the battery and the scales of voltage and current involved, these types of accumulators are divided into smaller sections called modules. Those modules are interconnected in series and in parallel to achieve the desired values for current, capacity, voltage and energy deployment.

The main task is to find the cell type that is going to be used in order to match the specifications. The team got a sponsorship to purchase the LG HE2 Li-ion Battery Modules [15]. They are modules of 7 independent cells that generate 3.6 V DC in its terminals and store 75.6 Wh of energy in each module. (See Figure 9 to better understand the different types of cell modules available)

FIGURE 9. LG H2 LI-ION BATTERY MODULES. (IMAGE FROM [15])

To achieve the theoretical approximation the final design of the Horizon's 20 battery pack is made of 96 modules connected in series. Knowing that each module has got 7 cells connected in parallel, finally the team will be using a total of 672 Li-ion cells. This gives a total voltage of 345.6 V, a final capacity of 21 Ah and an energy capacity of 7.26 kWh. This way all the values are very close to what we were aiming for.

Inside the battery pack and connected to each module we can also find a Battery Management System so called BMS. It is a circuit in charge of ensuring the good state of the cells by monitoring certain parameters that can be critical. The main parameters to oversee are temperature and state of discharge. When one of the parameters is out of its proper range the system can act and correct it by opening the circuit or equalizing the discharge of each module.

1.4. PROJECT STATUS

After the last three years, the development of UTRON's first prototype is starting to come to an end. During the first year the team focused mainly on the creation of the project and the CAD design. Finishing the second year, the goal was to complete the overall design of the car and start the production. In this third year the manufacturing part is still ongoing but we already want to start thinking about the setup. (In Figure 10 the current status of the manufacturing process is presented)

FIGURE 10. HORIZON 20 MANUFACTURING PROCESS. (IMAGE FROM [OWN ELABORATION])

The setup of the electric part relies on the powertrain and here is where the dissertation project starts gaining sense. The time has come to start assembling the powertrain components, setting up the correct parameters to ensure a good communication between them and this way bringing to life for the first time to the Horizon 20. Besides that, the idea is to capture and visualize the data obtained through a self-developed application.

2. OBJECTIVES

This project is framed in the development of the powertrain of the first UTRON vehicle for the Formula Student competition. In concrete, this work focuses on the real-time monitoring of data captured by the motor controller and its visualization on the driver dashboard. As a result, three main objectives are distinguished:

1. Setting up the motor controller.
2. Logging of motor data and its storage in a database for later analysis.
3. Development of a dashboard able to show the main parameters in real-time.

Describing each of these points in detail, the project starts with the powertrain. The main specific goal in this section is to set the controller up so data can be extracted. This requires a previous study of its functioning and its connections. A second specific goal here is to also obtain the material needed to perform the assembly and connection with the engine and the computer. In fact, a small testing bench for the powertrain is wanted in order for the team to check the components before assembling them into the car.

The second main objective is to create a database with the information from each run that the team can analyze afterwards. The idea is to select the most important parameters monitored by the electrical system and the controller and log them all through software programming. To ease the data acquisition and interpretation the file formats in this part will be key.

The data collected through the previous objective is expected to be shown to the driver in real time. That falls into the third main objective of this project. In motorsport competition this is achieved through a dashboard located on the front hoop of the car. The steps prior to creating this user interface will be to define the hardware and the software that suits best for the job. Also, a layout design of how the information is going to be represented is to be done. Last, the manufacturing and the packaging of this system into the car will be tackled during the process.

Even though the project has been described as three, seemingly standalone, objectives, in fact they are all expected to work as a unique system. It means that after the development of all the parts there will be a final challenge, which is the integration of all of them together. Moreover, the system must be autonomous enough that, when the car or the testing bench is working, no human action is required to set it up or extract the data, just running the application.

Last but not least, in the general context of the topic, this work aims to fulfill a further non-technical goal. Indeed, knowing that the work done is part of the Horizon 20, besides the push that this can give to the team development wise, an objective is also the elevation of the UTRON Racing brand and the team’s work through all the different stages of this thesis.

3. STATE OF THE ART

The step prior to the pure development of the project is knowing what is the current state of the field in which the dissertation is based on. This places the work in a certain context and can help to find the direction of the project considering the new tendencies. For this particular part, research was done regarding the electric powertrain technology and the dashboard technology always looking at the motorsport world.

3.1. ELECTRIC POWERTRAIN TECHNOLOGY

The state of the art of electric powertrains in motorsport has seen significant progress in recent years as electric vehicles (EVs) and hybrid technologies have gained prominence [16].

Advancements in battery technology have significantly improved the energy density and power output of electric powertrains in motorsport. Lithium-ion batteries with high discharge rates and quick charging capabilities are commonly used, providing the necessary energy for race durations.

Electric powertrains often incorporate regenerative braking and energy recovery systems. These systems convert kinetic energy generated during braking into electrical energy, which can be stored in the battery and later utilized for additional power during acceleration.

Electric vehicles also offer instant torque, even at low rpm, providing rapid acceleration from a standstill. The high torque output allows for quick launches and overtaking maneuvers, resulting in exciting on-track action.

Motors used in electric powertrains have become more compact, lightweight, and efficient, thanks to the developments in motor technology. Additionally, power electronics and inverters have improved, enabling better motor control and energy conversion.

These components require efficient thermal management systems to maintain optimal operating temperatures. Advanced cooling techniques, such as liquid cooling or direct immersion cooling, are employed to ensure the longevity and performance of the powertrain.

FIGURE 11. FORMULA E GEN 3 CAR DESIGN. (IMAGE FROM [16])

And the competition that brings all this together is Formula E, the world's first all-electric single-seater racing series that has been at the forefront of electric powertrain technology in motorsport. The series has seen continuous development in battery technology, power density, and efficiency, allowing the cars to achieve higher speeds and longer race distances. Always knowing that this technology can be added to road cars in the near future. (Figure 11 shows the last Formula E design that includes all the advancements previously described)

3.2. DASHBOARD TECHNOLOGY

In motorsport, the dashboard plays a crucial role in providing real-time information to drivers, enabling them to monitor various parameters and make informed decisions during races. Dashboard technology nowadays encompasses many different features and trends [17].

Traditional analog gauges have largely been replaced by digital displays, which offer more flexibility in terms of the information that can be displayed. These displays can show various metrics such as speed, RPM, gear position, lap times, fuel levels, and engine temperature.

Also, modern dashboards allow drivers to customize the information displayed based on their preferences. They can choose which metrics to prioritize and how they want them presented, ensuring the most relevant data is easily accessible.

The most advanced dashboards often integrate telemetry systems, collecting and displaying real-time data from various sensors on the car. This data can include tire pressures, suspension settings, brake temperatures, and aerodynamic performance, allowing drivers and engineers to make data-driven adjustments during races. They feature extensive data logging capabilities as well, capturing a wide range of information during races. This data can be analyzed after the race to evaluate performance, identify areas for improvement, and develop race strategies.

The same cockpit may include built-in communication systems that enable drivers to receive instructions from the pit crew, relay information, and receive updates on race conditions, such as weather changes or safety car deployments.

In addition, motorsport dashboards have started incorporating augmented reality (AR) overlays. AR can project extra information directly onto the windshield or visor, such as racing lines, optimal braking points, and virtual competitor positions, providing drivers with enhanced situational awareness. So far, it has been only tested in teams' racing simulators like the one captured in Figure 12.

FIGURE 12. FORMULA 1 TEAM RACING SIMULATOR. (IMAGE FROM [28])

In the future it is expected that certain forms of motorsport integrate video feeds from external cameras, including front-facing, rear-facing, and side-view cameras in the dashboards. This would allow drivers to have a comprehensive view of their surroundings and make more informed decisions.

It's worth mentioning that the specific features and technologies employed in motorsport dashboards can vary depending on the series, teams, and budgets involved. Furthermore, advancements in areas such as artificial intelligence, machine learning, and connectivity may continue to influence the future development of dashboards in motorsport.

4. PROJECT DEVELOPMENT

Once the objectives and the roadmap to follow have been defined, the following sections will be devoted to the details of the project development. Starting with the controller and ending with the graphic interface, all the steps in between are described in order to understand the results that will be seen later.

The work starts focusing on the motor controller PM100DX. It is the component that will help to obtain all the needed data. That is the reason why in the first place it is important to understand its functioning. This can be done through the different documents that its manufacturer CASCADIA MOTION provides [18][19]. Not only is the hardware explained but also the software that runs it together with the engine.

4.1. CONTROLLER RESEARCH

As previously described, the controller operates on the power electronics and the control electronics. Both are directly related taking into account the 7 typical wirings that the inverter admits. (Figure 13 contains a brief map of these wirings and their importance)

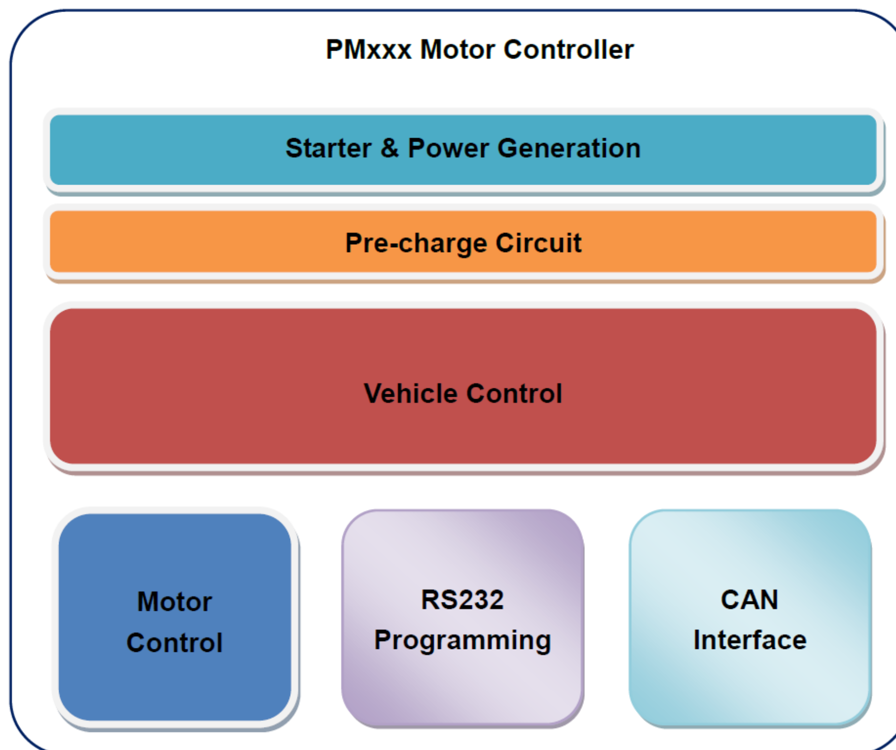


FIGURE 13. PM100 TYPICAL WIRINGS. (IMAGE FROM [18])

First, we can find the power supply wiring. Apart from the power it receives from a high-voltage DC source that must be transformed into AC, the controller also needs 12V to run its internal circuitry. This can be configured in two different ways, directly connecting the controller to the low voltage system of 12V, or adding a switch in between to be able to manage the ignition procedure.

However, it is also prepared to implement a pre-charge circuit in the input of the high voltage part. It is composed of two electrically controlled relays, two fuses[11] and a resistor. The relays are connected and managed by the controller through the pins that it contains. The overall goal of the pre-charge circuit is to limit the initial inrush current, due to the charge of the input capacitors of the controllers, thus protecting them. Additionally, the fuses protect the powertrain in case a peak of current arrives to the controller and the engine. If this happens the fuses brake, the controller detects that the two branches aren't connected and opens the circuit acting over the relays. Figure 14 shows the actual circuitry as reference.

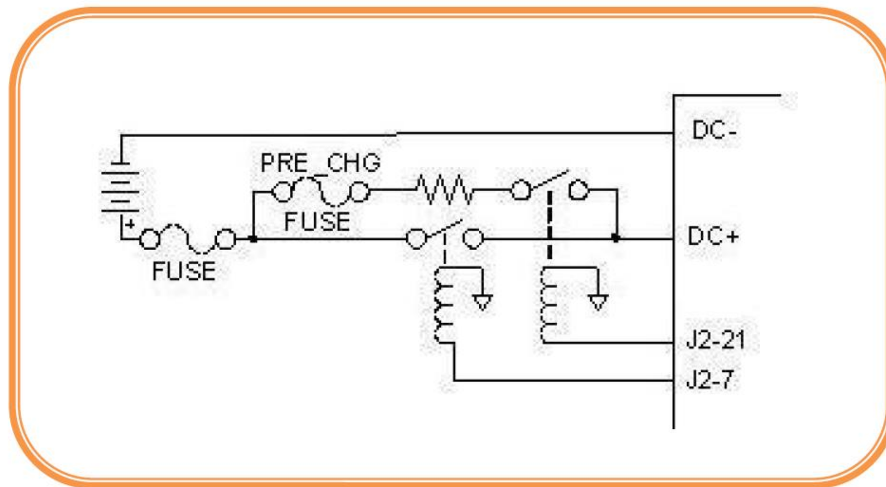


FIGURE 14. CONTROLLER'S PRE-CHARGE CIRCUIT MORPHOLOGY. (IMAGE FROM [18])

The third attached circuit that the inverter can manage is the vehicle control. Again, connecting the correct signal pins and thanks to a potentiometer that will regulate the throttle and three switches that will act on the braking and the direction of the movement the vehicle can be controlled. It is important to mention that there are also several pins reserved for the braking in case it wants to be managed gradually with a potentiometer.

For the motor control the PM100DX has 13 pins available in addition to the three high-voltage connections that will be used. Some of them are for the use of an encoder and some of them are for a resolver. These two devices help the controller in the task of knowing the exact position of the motor axle. With this information the controller can manage the best way possible the modulation of the AC signal. The EMRAX 208 is equipped with the second option, a resolver, that is why the pins to be used are the ones shown in the following figure, Figure 15.

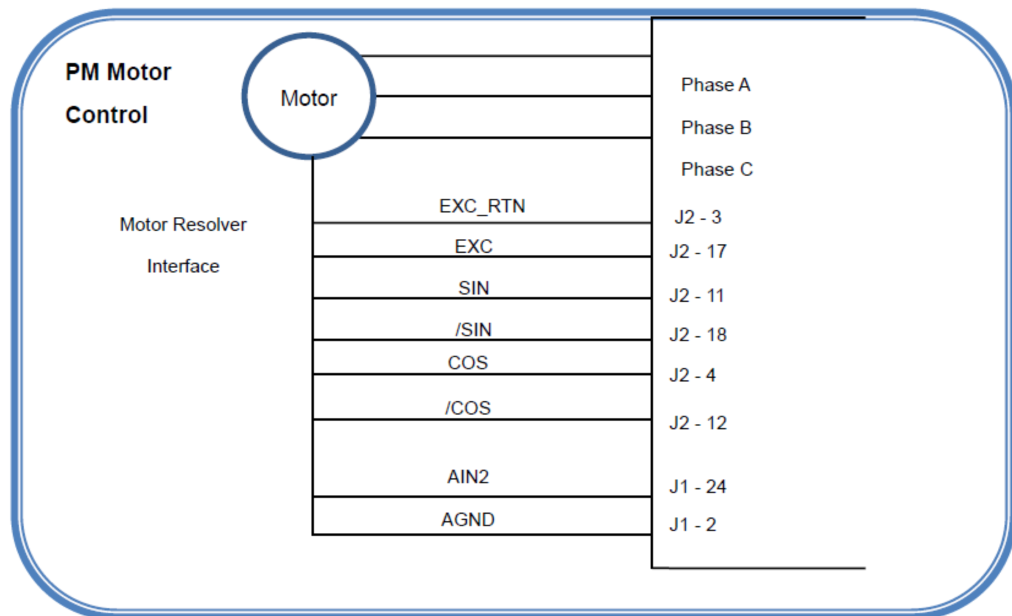


FIGURE 15. EMRAX MOTOR CONNECTIONS. (IMAGE FROM [18])

In addition, there is one RS-232 serial interface with EMI[12] filtering. This port can be used to set up and tune the controller, as well as to upload controller software updates from a PC. The manufacturer offers a simple serial user monitor that runs on the PC to allow changing parameters. The drive can also be placed in a data-logging mode and, used with a PC or other serial device, the unit broadcasts at 3Hz datasets of a number of parameters that allow performance and energy consumption data to be gathered in real time.

And last, it is important to mention that all this exchange of data and configuration can also be done through the two CAN[13] networks that the inverter has, CAN A and CAN B. Only CAN A is active, CAN B is reserved for future use. The CAN protocol conforms to CAN 2.0 A using 11-bit identifiers. The default bus speed is 250kbps and every message has a data length code (DLC) of 8 bytes.

For the purpose of the project, the focus will go on the aspects that interfere with the vehicle control, motor control and RS232 Serial Interface, which will be the data source.

4.1.1. Controller connections

For data applications the unit is designed with 58 different pins. These pins are distributed along two AMPSEAL[14] connectors, one with 23 pins and a second one with 35 pins [20][21][22]. AMPSEAL connectors are designed for cable-to-board harsh environment applications. (The shape of the connectors is shown in Figure 16 and 17) The pre-assembled receptacle housing connector used in this case features built-in contact sealing to eliminate individual wire-sealing grommets, while an integral interfacial seal protects mated connectors.

FIGURE 16. 23 PINS AMPSEAL CONNECTOR. (IMAGE FROM [21])

FIGURE 17. 35 PINS AMPSEAL CONNECTOR. (IMAGE FROM [22])

To tell apart the different pins, the controller documentation comes with a list that matches each pin with its circuit or purpose. During the development of the dissertation not the 58 pins will be used. First, because some of them are redundancy pins to have a second connection in case the main one fails. Second, because, as mentioned previously, there are some others that take into account applications that step outside of the project's scope.

In fact, the pins that will be useful for the task are those that are related to the engine control, the controller parameters, and the data acquisition. In other words, the throttle, the brake, the resolver, the high voltage circuit state, the serial port, and the power pins.

4.1.2. Material purchasing

Bearing in mind what was going to be monitored and how it worked, the next step was prepared. To carry out the connection there are some materials or small components that need to be acquired. When the team bought the controller, nothing came together with it. Therefore, a small process of purchasing was done before starting the assembly.

In this purchasing order the following material was obtained:

- Two AMPSEAL connectors required to adapt the different wiring to the signal inputs of the inverter.
- 100 AMP socket connectors for the proper position of the smaller wires into the connector.
- 24 meters of eight-way cable meant to connect all the controller pins.
- A DB-9[15] female plug to connect the inverter output with the serial port.
- An adaptor DB-9 male plug to USB since most devices no longer have DB-9 input.

The rest of the material used, referring to basic electronics components such as potentiometers or common wires, was obtained from the university's automotive laboratory or the team's inventory.

4.1.3. Assembly

With the material's aspects solved, the connection and assembly of the different parts was started.

The elaboration goes from the very simple and small objects to the interconnection of the bigger components. That is why the first thing done was the cutting of the wires. The 24 meters of cable were trimmed in 7 shorter cables. Knowing that each of the cables had eight-ways inside $7 \cdot 8 = 56\, wires$ makes the 54 useful pins out of the 58 pins are covered.

After that, each of the smaller wires had to be crimped with its AMP socket using a special crimper. Once they had the corresponding termination, they were introduced in the AMPSEAL connectors. (Figure number 18 can be checked for further information) To recognize each of the cables and their matching pin a color codification was established. In Appendix B, a supplementary table on the color codification is presented.

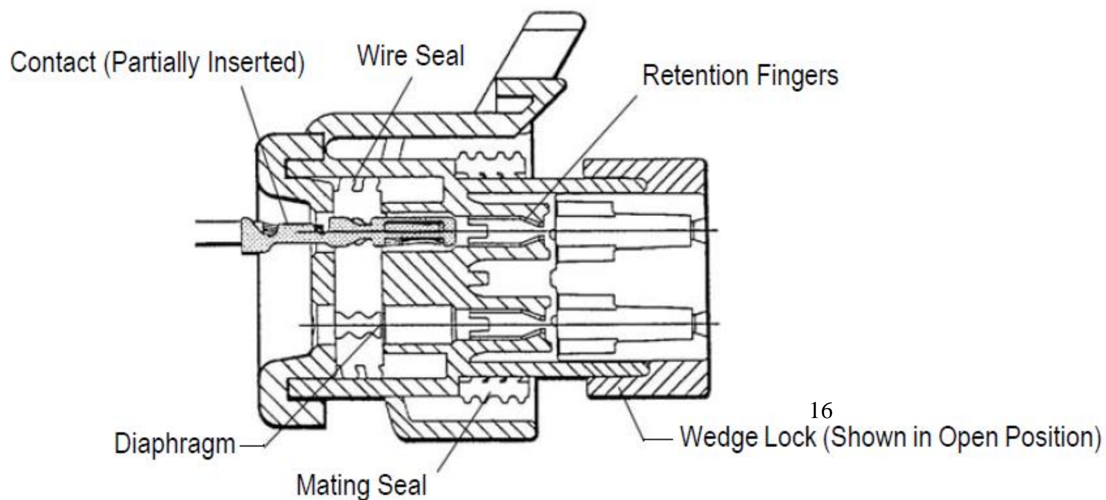


FIGURE 18. AMPSEAL CONNECTORS CONNECTION PROCEDURE SCHEME. (IMAGE FROM [20])

As the last part for the AMPSEAL connectors, the only thing left was to introduce them into the controller and ensure that the assembly was secure with no possibility of falling off.

Before carrying out the final connection all the components have been placed and attached on top of a wooden board as a testing bench. Each component and wire has its own place allowing them to be easily recognized and in general the model is very handy in terms of use and transportation. (In Figure 19 an image of the final model has been captured)

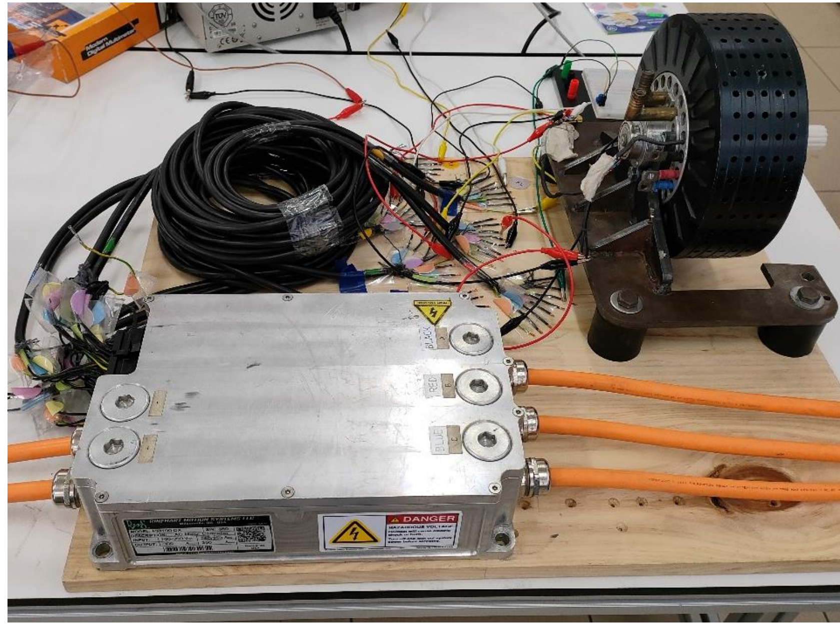


FIGURE 19. HORIZON 20 POWERTRAIN MODEL. (IMAGE FROM [OWN ELABORATION])

Nevertheless, the connection of the terminals was still to be done. To start with the simple configuration six wires were used for the engine resolver, two for the engine temperature sensor, three for the accelerator potentiometer, three for the brake potentiometer, one for the forward direction switch, another one for the reverse direction switch, two for the power supply and three for the serial port.

The final three wires that go to the DB-9 had to be welded to the adaptor. Following, this connects with the DB- 9 - USB serial port adapter in order to receive that data in a device such a PC or a microcontroller.

4.1.4. Controller software

The PM100DX inverter has its own software that can be downloaded from the developer's webpage. It is called RMS[17] GUI for Graphical User Interface and it is a Windows application developed by Cascadia Motion based on C2Prog programming tool. This application communicates over an RS232 port [23][24].

The primary purpose of this application is to be able to monitor a specific set of parameters in real time. However, the application also provides the ability to program certain EEPROM[18] parameters. The set of EEPROM parameters needs to be modified based on each motor and other system set up by the customer. EEPROM parameters must be programmed correctly before the controller is operated.

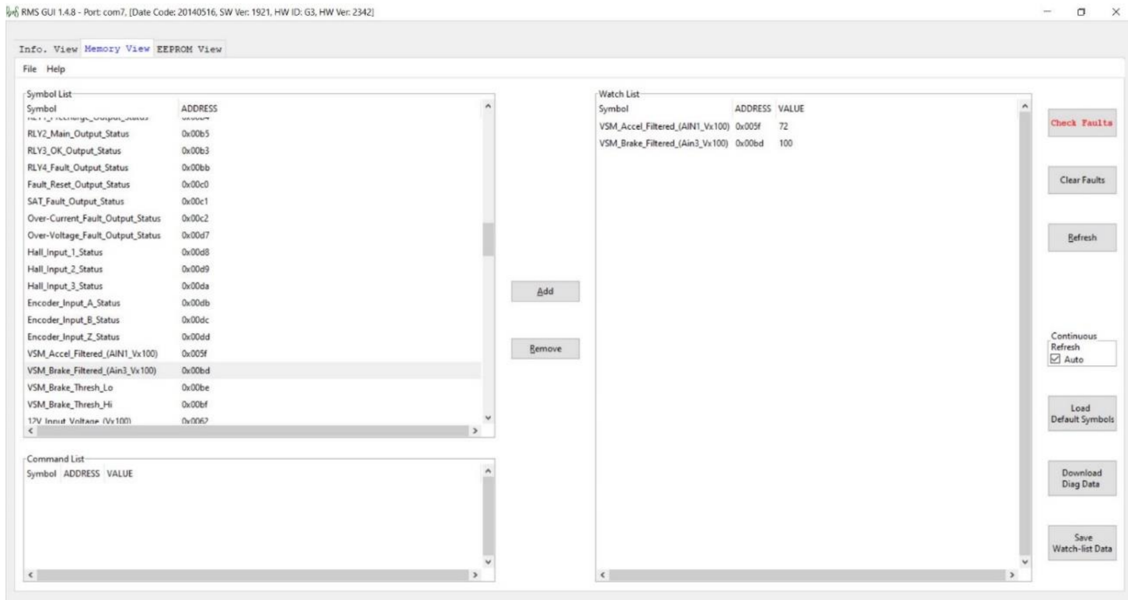


FIGURE 20. CONTROLLER SOFTWARE MAIN WINDOW. (IMAGE FROM [23])

As seen in Figure 20, with a simple interface the software allows the user to upload a default symbols file and act over two main windows. The memory view is used to check on real time data and the EEPROM view is used to check and modify the configuration parameters.

Even though the software has been used in the first stages of the project and the connection phase, the simple features that it offers fall short on the final goal of the dissertation. It is a good tool to start but only useful for very simple applications. That is the reason why another code has been developed during the work process with the same philosophy but greater possibilities.

This new code is developed in Python language using Spyder as its scientific environment. The decision was taken regarding the multiple libraries available and the mass of possibilities that it offers, among them its availability in several microcontrollers and the easy replication of the codes.

4.1.5. Data Acquisition Program

The first part of the code will perform the task of acquiring and deciphering the data. Since the information is coming from the serial port it is very important to use the python library “serial” to be able to work with this data. From there, the small code that will allow the rest of the project to manipulate the information of the controller is divided in four main tasks [25].

First the class “Record()” has been created. It contains several functions to divide, organize and translate the different lines that the unit is sending ASCII[19]-encoded. This class is later called every time a line is read but in this part is only defined for future use.

With the command “serial.Serial()” the serial port is set to match the standards in which the inverter is sending the messages. With this function the port name, the baud rate, the byte size, the parity check, the number of stop bits, the timeout, the software flow control and the hardware flow control can be configured. In the PM100DX datasheets these values are specified as shown in Figure 21.

Baud Rate	57600
Parity	None
Data Bits	8
Stop Bits	1
Hardware Flow Control	None

FIGURE 21. CONTROLLER’S SERIAL PORT SETTINGS. (IMAGE FROM [24])

The following lines of the code are used to make sure that, when the SCI mode starts, the buffer is empty and ready to receive a new complete line. This way all the information that is registered is fully comprehensive and it can not cause any issue to the data format defined previously. Once this is done the serial communication interface starts by writing into the window the symbol ‘+’ and pressing enter as the documentation explains. To make this procedure automatic the following line is implemented into the code: “ser.write(b"+\n\r")”. Using the same command, the SCI mode must be stopped after the code is run and before the serial port is closed.

Last, the loop to read not only one line of information but a series of lines is developed. This loop can be managed by a certain number of repetitions with a “for in range” or with a “while” depending on the value of a variable. The iteration consists of reading lines if there is any or starting the SCI mode if there isn’t. When the line is received it goes through the previously defined class called “Record” to modify its format and it is later printed in the console. The full code can be found in the ANNEX C.

This way, what we achieve with this code when it is completed is a flow of data translated and printed in the console. If it is printed in a window, it means that it can also be manipulated and used for different purposes. Included in these purposes are its registration and its graphic interpretation, which are the main goals of the project.

4.2. DATA LOGGING

The reason for acquiring and setting a structure for the data is easing the post processing work. Taking into account the output of the program explained before, the development of a data logging system starts gaining sense. In this second part the lines received are registered into a document, increasing the capabilities of the team to analyze the performance of the car and its elements. In this case the code focuses on the components that the inverter is monitoring but it can be extended to different systems.

4.2.1. Data Selection

The controller sends twenty different parameters through the serial port. The twenty parameters can be consulted in ANNEX D. All of them are useful but they don't have the same relevancy. For that reason, only eleven out of the twenty parameters will be taken into account. From each serial port message, the data registered will be:

1. Time
2. Throttle value
3. Brake value
4. Motor torque value
5. Vehicle torque value
6. DC voltage value
7. DC current value
8. Motor speed value
9. Motor voltage value
10. Controller temperature value
11. Motor temperature value

Apart from these eleven values a twelfth one has been added due to its importance. This is the vehicle linear speed or commonly known as the speed. It is not a parameter that comes out of the inverter because the unit has not enough information about the characteristics of the car such as the transmission ratio or the tire diameter, but it is an interesting magnitude to be measured and showed to the driver at each moment. And for the programming part it only means to add an extra variable with a division over the engine revolutions per minute. For further information regarding speed conversion, please refer to ANNEX E.

4.2.2. CSV and Openpyxl Libraries

Once the data that wants to be processed is defined the methodology of how the logging is going to be performed is also chosen. The same task can be performed in many ways when programming but in this project, it has been carried out using two standard python libraries that allow the user to manage data easily.

In the first place, the idea is to translate the data into CSV[20] format. It is a very simple and common way to write, read and manipulate information. This can be done through python in a very efficient way with the library called "csv" that is installed by default. Moreover, the information can be flushed very often, and it can be read with any ".txt" files' reader at any time [26].

The CSV storing is a safety step prior to the processing of the data thinking about how the car will work. The CSV format does not need the code to finish running to save the changes compared to other complex applications. This means that if, during the competition or testing any issue makes the controller stop working, the last information will still be saved and the team will be able to access it to know what happened. Otherwise, the file would not be saved or would be damaged and the information would be lost.

In second place, the csv file can be transformed into a different format through software. The csv format is very handy, but it does not allow to make a proper differentiation of the data to ease its visualization afterwards. To do so, the library "openpyxl" is a good option

[27]. It is also a standard library that creates and configures excel files through python programming. With this step, the data logging program takes the csv file recently written and creates a structured excel file in the end. Once the document is created all the different features of an excel worksheet are available. The user can create graphs, remark certain values, compare cells…

4.2.3. Code explanation

To speed up the development of this code avoiding the connection of the controller every time a test was performed, an example line was used. This line has the same structure as the lines that the inverter sends, and it is a good representation of what will occur when this program joins the serial communication code explained previously.

The data logging program starts initializing all the libraries that will be called later. Then the example line is defined and divided. This way each of the values of the line is a different element of the list. With the command "with open ('CSV file.txt', 'a', newline='') as file:" a text file is created and the csv format can be configured as desired. Also, the lines are being appended to the file until we press the esc button that will break the loop.

After the CSV file is written the openpyxl library takes over. It starts by opening a workbook and structuring it. With the command "wb.create_sheet" a different sheet is created for each of the parameters that are going to be registered. The first sheet is reserved to store traceability information such as the time and the day the file is being created.

When the sheets are created it is easy to refer to them and add information to specific cells. This feature is used to write some headings on top of each of the sheets. The standard format of the sheets will be two columns, one with all the time lapses and another one with the value of the parameter at each time stamp.

After the preparation of the final document all the information from the CSV file is imported. Thanks to the csv library the file is read and each of the elements of the lines is labeled according to the information that they contain. A middle step is performed to create lists that join the time stamp and the value of the parameter. This part is also used to divide the information in order to obtain the real magnitude.

Finally, when the "esc" button in the keyboard is pressed, the lists with the pairs of time and value are added to the correct sheet of the excel. This time the values are not added to a specific cell because the loop would not allow the user to know which is the cell number. Instead, the lists are added at the first blank row of the sheet with the function ".append".

When the importation has ended the file can be saved with the desired name. For UTRON purposes it is manageable to have the date and the time in which the file is created as its name. On the one hand because it makes them easily recognizable and secondly it changes its name automatically every minute. Otherwise, if the file had always the same name, it

would be replaced and the previous file would be lost. The full code can be found in the ANNEX F.

4.3. DASHBOARD

In the automotive industry a dashboard refers to the panel that contains the instrumentation and controls for a vehicle. It is typically located in front of the driver and provides important information that he needs while operating the vehicle. This is not only used in road cars but also in racing cars and other special machinery. It is an easy way to visualize real-time data and see what are the consequences of the operations that are being performed. Nowadays the best way to do so is through a screen.

Since the team is treating data and analyzing it, the idea to implement this type of technology to the Horizon 20 was seen with good eyes. Therefore, the information that is being logged is at the same time displayed in front of the driver. The main purpose is the identification of critical situations and giving the driver the capability to act as soon as possible to avoid further issues. For instance, if an element is overheating, the driver can modify the driving style to reduce that temperature or stop the car if that is the best solution.

Moreover, this add-on can be very helpful when preparing the car because it allows the team to see if everything is working properly. If a key component is not sending any signal to the dashboard, meaning that it doesn't appear reflected on the display, it warns that a problem occurs; component failure, bad assembly, wrong connections…

4.3.1. Dashboard interface design

The design of what appears in the dashboard and how it is displayed is ruled by two factors. From one hand, based on how other motorsport teams do the job: Formula Student, Endurance, Formula 1… and from the other hand which parameters the team is able to show or wants the driver to see.

FIGURE 22. FORMULA 1 DASHBOARD. (IMAGE FROM [17])

The dashboard usage is nothing new in the motorsport field. All the competitions find a way to inform the driver about the state of the car while driving. Designs go from the

simplest LED indicator to the multi-page screen concept seen in Formula One [28][29]. (Williams Formula 1 Team concept is shown in Figure 22) This last one takes advantage of the colored screen to show hundreds of data and messages with different colors of numbers, letters and bars.

However, the most generalized design in Formula SAE nowadays uses alphanumeric LCD[21] displays [30]. They can only show a limited amount of information at once and the format must be numbers or letters. In contrast, they allow the teams to keep the driver updated in a way that is cheaper and easier to implement.

In the design of UTRON's dashboard nine out of the eleven parameters that are logged from the inverter are shown. All of them except for the time and the vehicle's torque which give no relevant information to the driver. Instead, the linear speed and the temperature of the wheels and the accumulator are added.

With these two ideas in mind the layout of the final display can be seen in Figure 23. The layout is the background of the interface, and it receives this name because it is always static and defines the overall structure. From there, all the information that varies and is being updated is drawn on top of it, but the background never changes. The picture has been modelled using the Adobe Photoshop application [31].

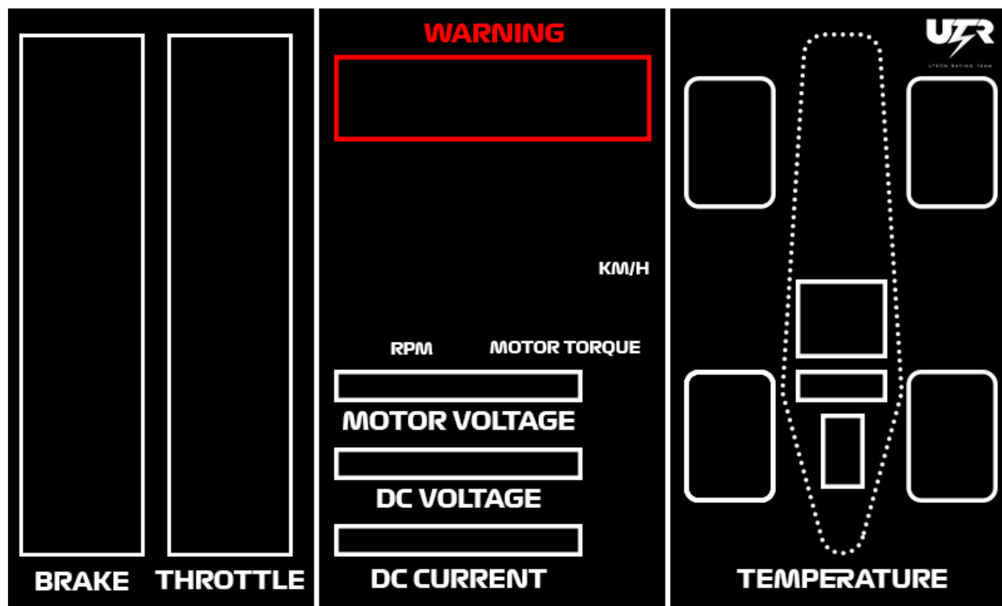


FIGURE 23. UTRON'S DASHBOARD LAYOUT. (IMAGE FROM [OWN ELABORATION])

Starting from the left side two bars recreate the percentage of throttle and brake pressure that is being applied. The same method is used to notify about the motor voltage, DC voltage and DC current in the bottom part of the middle section. Some room is also reserved to indicate the value in numbers on the right side of the bars.

The upper middle part is intended to show the most significant data. In first place warning messages that appear in case of emergency, and, right below them, the linear speed, the

revolutions per minute and the motor torque. All the speed and torque values are represented in numbers and not bars.

The third section on the right is designed to give information about the components' temperature. A color scale is defined for each range of temperatures and this color fills the component in the screen. So far, the team is only receiving information about the heat in the engine and the controller, but, thinking ahead, the temperature of the wheels and the accumulator will also be monitored, that is why they appear in the layout as well.

4.3.2. Dashboard hardware

The dashboard itself is formed mainly by five different components: the microcontroller, the screen, the connector, the housing and the cockpit panel. Also, some screws and nuts are used to assemble together some parts.

Firstly, the microcontroller in charge of the processing is a Raspberry Pi 3 model B+ [32][33]. Despite the choice, a Raspberry Pi 4 could also be used. The selection of the motherboard was based upon the high processing capability, the ease of connection, the Linux[22] OS and the reduced volume. These factors stand out in the Raspberry compared to other microcontrollers available.

Secondly, the monitor in which the information is shown is the ELECROW 5-inch Touchscreen [34]. It is an LCD display with a resolution of 800x480 pixels and compatibility with Raspberry Pi 3 and 4 through HDMI[23] connection. In terms of dimensions and performance it is the same as the ones used in high level competition.

To connect the two previously described parts there is an small HDMI connector. It is designed in a way that fits perfectly with the microprocessor and the screen using the least space possible. It is an add-on provided by ELECROW[24] when the monitor is purchased and very useful in the project's purpose.

Apart from that, a self-made housing for both the display and the Raspberry has been designed and it has been 3D-printed in the university's facilities. The development took into consideration the dimensions of the assembled parts, the needed connections, the way it was attached to the panel and the cooling. (Figure 24 contains the CAD design look of this part)

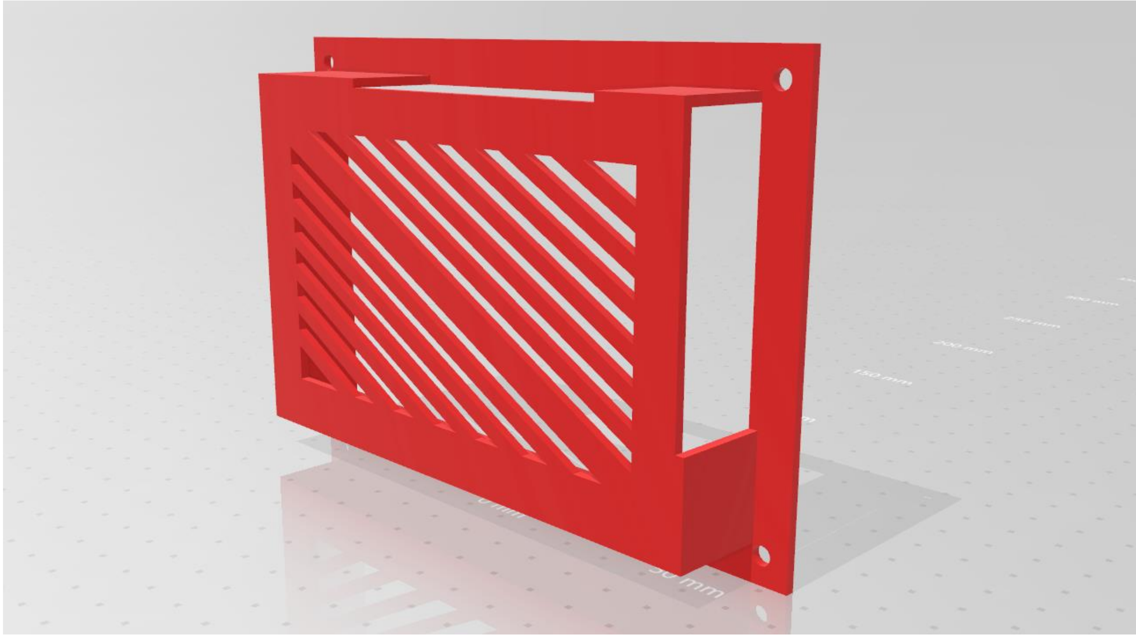


FIGURE 24. DISPLAY HOUSING CAD DESIGN. (IMAGE FROM [OWN ELABORATION])

With the attachment goal, the housing has got four holes in the corners. Thinking about the connections, some openings have been added to make sure the motherboard was not isolated and it could be reachable. Finally, a diagonal grid was implemented in the back side to allow the heat to get out and cool down the electronics. In the ANNEX G, additional details on the general dimensions of the housing are provided.

Last, the cockpit panel has been manufactured. The contour follows the surface of the front hoop and the bottom part is limited by the steering system. In the middle part a cut is performed to fit the touchscreen. It is made out of a aluminum composite material with a thickness of 4 mm. (Figure 25 contains the main dimensions of this dashboard panel)

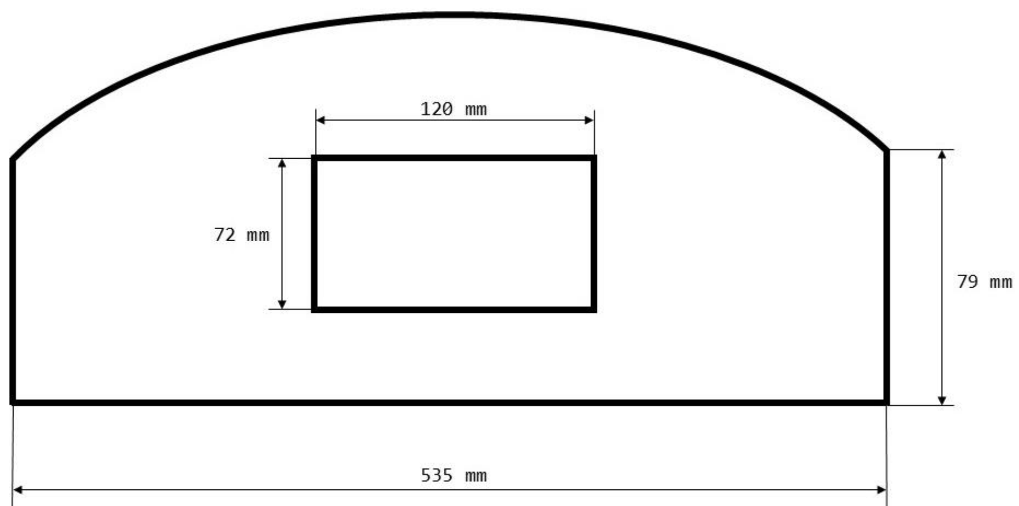


FIGURE 25. DASHBOARD PANEL GENERAL DIMENSIONS. (IMAGE FROM [OWN ELABORATION])

When assembling the five dashboard parts the following process must be followed. The first step is to connect the pins of the Raspberry with the ones that match perfectly with the screen. Then, when the right position is ensured, the HDMI connector can be inserted on top of both. Once this is done the set can be fitted in the housing and the housing can be screwed in the panel. (The final result of the assembly is presented in Figure 26)

FIGURE 26. DASHBOARD FINAL ASSEMBLY. (IMAGE FROM [OWN ELABORATION])

At this stage, the panel would be assembled but in order to install it and make it work some further steps are to be taken. The microcontroller must be configured with the Raspbian OS and connected to 5 V power supply. The serial port of the controller must be connected through one of the four USB ports. In the end the overall system is attached to the front hoop and the lateral tubes of the chassis. In the case of testing situation, it can be attached to other structures.

4.3.3. GTK library

Graphical User Interfaces are computer programs that enable users to interact with electronic devices through the use of symbols, visual metaphors and pointing devices. It means the data is translated providing a more intuitive and user-friendly look. This type of programs can be also created through the same Python language used for the previous parts of the project. There are countless libraries and frameworks that allow this, but the ones that have been considered are the following three:

- *Tkinter*: it is the standard GUI library for Python and comes pre-installed with most Python distributions. It contains a range of pre-built widgets and classes that allow the programmer to create and manage graphical elements, such as windows, buttons, labels, text boxes, and more.
- *GTK*: referring to GIMP[25] Toolkit due to its beginnings, it is a popular open-source library primarily used for creating graphical user interfaces in various programming languages, including Python. It provides a comprehensive set of tools, widgets, and APIs[26] for building cross-platform applications.

- *PyQt*: it is a powerful Python binding for the Qt framework, which is a widely used and comprehensive GUI toolkit. It enables developers to create cross-platform applications as well with rich graphical interfaces and a wide range of features supported by various multimedia elements.

After evaluating the different options, the GUI library chosen was the GTK, more precisely, the GTK 3 version [35]. The strengths of this library that made the difference compared to the others are, in first place, its direct compatibility with Raspian (Linux) which is the OS that will be used in the display. The fact that it is an open-source library also makes its community active and the documentation is accessible. You can be guided by other people's examples or even ask about your code struggles. Furthermore, GTK provides an extensive widget library that can be integrated in many different programming languages and frameworks. It means that if the need for changing the coding environment within the team arises the same library could still be used. The same applies if the program wants to be implemented in a project with a different language.

Focusing on how this library will fulfill the needs of the dashboard design the very basics are broken down [36]. In the very beginning, the previously designed image is expected to be displayed as a background. On top of it the information is wanted to be drawn in the shape of rectangles and labels with several colors. And for the project purposes mentioned the Cairo context will be a very helpful tool. Cairo is a powerful 2D graphics library that GTK uses for rendering and creating visual elements [37].

For the background part, the command "Gdk.cairo_set_source_pixbuf()" followed by "cr.paint()" will display the previously saved image as a background. The "cr.rectangle()" function allows the creation of any type of rectangle with a series of given coordinates. If changing the color is desired, it can be done by writing "cr.set_source_rgb()" and indicating the RGB[27] combination values of the new color in between the brackets. Last some text can be inserted by using the functions "cr.text_extents()" and "cr.show_text()".

These few commands can make the work of drawing the dashboard by themselves and they will be repeatedly used during the development of the code. Apart from that some set up commands and basic programming will be used and are described in depth in the dashboard code explanation.

4.3.4. Dashboard code explanation

The first thing that appears on top of the code is the importation of the libraries. The GTK libraries contain all the different functions needed to create and develop the Graphical User Interface. During the program they are called for specific purposes and if they are not previously defined in the code it means they are already created.

Once this is done, the next step is to create a window which will be the scenario or base where the user is allowed to place the different widgets. The window is defined as a class, and inside some standard functions are programmed. The standard functions are parts of the code that not only apply for UTRON's dashboard but most of the GUIs.

In this class the window's size is determined followed by the importation of an image as a window icon. The "do_key_press_event" is defined so the window can be closed by pressing a button when no other button is shown in the screen. It is related to the "on_destroy" function that shuts down the program. And finally, some visual aspects are programmed concerning the final purpose of the app. The window is expected to be covering the full screen with no other window or object above it.

A second class is defined after the main configurations have been performed. This one creates what in GTK language is called "DrawingArea". It is a widget that enables the user to start placing things and drawing as it was a canvas. Here is where UTRON's exclusive layout and the parameters selected are relevant.

In the setup method or "__init__" of "UTRONDrawingArea" the background image is imported so it is available from the very beginning. Also, the draw signal is connected to the drawing method which is the one that allows the display of information.

When in the drawing method the functions explained in the previous section are applied with some styling considerations. In first place the background is painted and rebuilt every time the function is called. From there the rest of the shapes, numbers and letters are defined to be drawn on top of it with the same structure.

This structure first sets the RGB color in which the object is wanted to be painted. Then the value of that parameter is read and scaled if needed. Last, the object is determined taking into account the value read before and it is displayed in the dashboard in a certain position. In this development part, the values are written by hand, and they are static, but it is important to know the range that the parameters can take in order to ease the transformation to the final dashboard design.

It is worth mentioning that for the information related to temperature the structure is a bit different. Instead of establishing a single color to the objects, a color code and a condition has been programmed. Depending on which range of temperatures the component is, the RGB value to consider is one or the others. The reasoning is that if it is in the range that is considered below the optimum temperature the color is blue. If it is between good temperature values, the color is set to green. If the temperature starts to be above the optimum levels the RGB value shows a yellow configuration. Last if the component is reaching dangerous values that can put the driver and the prototype at risk the color displayed is red.

Finally, concerning the integrity of the drivers, a set of warning messages have been pre-programmed to be activated whenever a problem occurs. The possible issues have been classified depending on its importance. They appear in the middle of the screen and inform the driver about the danger and how to act accordingly. For that reason, they are divided in two sections, the problem message and the action message.

Since all the functions explained were only defined and never run, at the end of the code they are called in order to start working. First, the window is assigned to “db”, second the drawing area is added to the window and then the code is run. For further information the full code can be found in the ANNEX H.

4.4. CODE INTEGRATION

After the different codes have been developed so they meet the expected features independently, they are to be integrated all together to work as one. Until this point each of the programs was tested by itself with specific values that were constant. Since the final goal is to have a non-static dashboard that redraws itself depending on the data obtained, the information must flow between the different codes. The SCI code must be connected to the Data Logger and at the same time to the Dashboard in order to have a dynamic system.

4.4.1. Threading

For that specific purpose Python counts on a library that can be used, it is called “threading” [38]. It is named after the seam concept due to its similar functioning. The “threading” in programming provides a way to create and manage lightweight execution units called threads within a single process. Threads allow concurrent execution of multiple tasks, where each task runs independently but shares the same resources as other threads within the process.

The library contains methods to manage threads, including functions to check if a thread is alive, to wait for a thread to complete its execution, to set thread names... It takes advantage of waiting moments to run other parts of the code. This allows for synchronization and simultaneity of tasks.

This tool is the one that will be used to assemble all the codes in the project and ensure correct functioning. In fact, it will allow concurrency between the serial messages reading and logging, and the dashboard drawing. But to do so, some changes must be performed over the different parts of the code in order to implement the “threads”. At the same time the codes have been refined deleting and improving parts and including the different steps in functions that help with the organization and the identification of sections.

4.4.2. Code Adaptation

In the first place the structure of the codes has been modified to ease the adaptation. The program is as well divided into three parts, but its content is quite different. One of them contains only the excel translation function. The second one includes the serial port reading, the CSV writing, and the threading management. Last, the graphical interface can be found but with some changes as well.

As for the excel part, the “UTRON_Excel”, the composition is the same as in the data logging code used for the development. The excel document is organized using the library “openpyxl” and the data is distributed in each correspondent cell. However, in this case,

it is all included in a function that is called in the end. Also, the CSV file is not created in this step yet, it is only read. Therefore, this widget will only work as a translator of information, from CSV file to an organized excel. That is because the part of the code in charge of actually saving the information is the second one with the serial communication.

The second widget has been named “UTRON_Logger” because it contains the basic functions to gather the information received from the serial port. It means the SCI and the logging coexist now in the same code. Apart from that, some new features have been added. In this new version the class “UtronLogger” creates a thread and a queue. The thread is managed by a semaphore or flag. The flag is the data flow manager, if the flag is deactivated the program can load the information to the queue. That is why the main part of the code is inside a “while not” loop ruled by this semaphore. The same way it was done before, the data recorded is being added as new lines in the CSV file. The last improvement is that the parameters the GUI will draw are already converted to the correct magnitude in a number of programmed properties. With this step the conversion in the GUI code is avoided.

In the third place there is the GUI program, also known as “UTRON_Dashboard”. It ends up being the only application that needs to be executed because in the very beginning it imports the other two as widgets, making all of them work as one. The variations considering the previous dashboard created are basically two. On one side the parameters are no longer an imposed value, they are a variable coming from the serial port. They are initialized in the “DrawingArea” class. The function “on_idle”, referring to the idle[28] state in the window class, is in charge of asking for the value of the parameter in the “queue” over and over again in order to redraw the dashboard. On the other side the other two codes need to be called in this one to make them work. That is why at the end there is a series of extra lines. First, they connect the queue in the thread to the dashboard. After that, the logging starts, giving the dashboard the first data to work with. When ready, the main loop is run and until the signal to stop is not sent the program continues working. If the “esc” button is pressed, the loop is broken and the logging and communication part stops. The final step is to call the excel function in order to save the information in this format. Once this is completed the program is over.

Last, it is important to mention that the two widgets can run on their own as well. At the end of each of their codes, they are executed inside a condition. This condition means that that part of the code will only be read if the code is part of the main section. This makes them work when executed separately and avoids the fact that they are run twice when they are imported into another code.

To better understand the program functioning refer to Figure 27 which contains a diagram with the different parts. Moreover, the code of the widgets and the code of the final dashboard application can be found in ANNEXES I, J, and K.

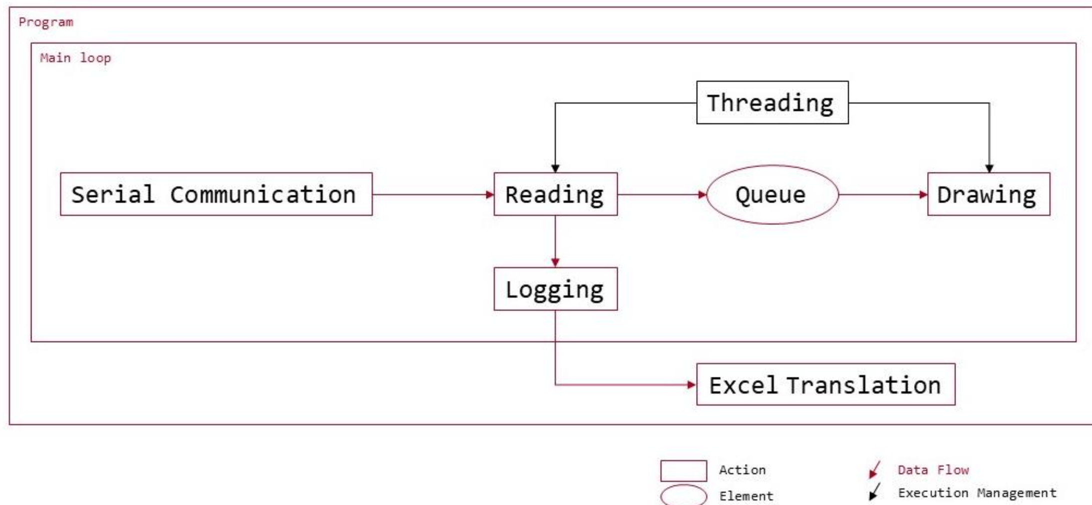


FIGURE 27. FINAL CODE DIAGRAM. (IMAGE FROM [OWN ELABORATION])

4.5. TESTING

As a final check of the performance of the system some tests have been carried out. These have helped to establish the process that the team and the driver must follow when executing the dashboard with its key steps and important tricks. Besides that, they have allowed to see if the expected results were obtained.

The program can be loaded on the Raspberry Pi memory as any other file. The team must make sure that all three codes are in the same folder so they can be imported with no further issue. From there, the "UTRON_Dashboard" file is to be run when the test or the competition event starts and the team nor the driver must worry about anything else until the end of the stint.

When the run is finished, and the car is back in the pits the team is now responsible for stopping the program and downloading the information that the system has logged. Inside the same folder where the application is, two new files appear, a CSV file and an excel file. The CSV is a temporary document used to load all the information on the excel file at the end. It is recommended to either change its name or delete it to avoid being mistaken by the system the next time the program is executed. As for the excel file, it will contain the information from the performance of the car to be analyzed afterwards. This second code won't be overwritten because it is labeled with the date and time of the run, and it must be stored. If the information is already saved the team can proceed with another run or shut down the microcontroller if it is not going to be used again.

During the testing, the dashboard part worked according to what it was programmed. Also, the information was being recorded successfully and the team could start analyzing some aspects of the breaking and the throttle despite the car not moving. For instance, we could differentiate between two different situations in the graphics created in the excel.

In the first place we simulated what a progressive start is. How the brakes aren't pressed can be seen in Figure 28, that is why the voltage signal stays low. And the throttle pedal is constantly increasing until it reaches the maximum value, which is almost 5 V.

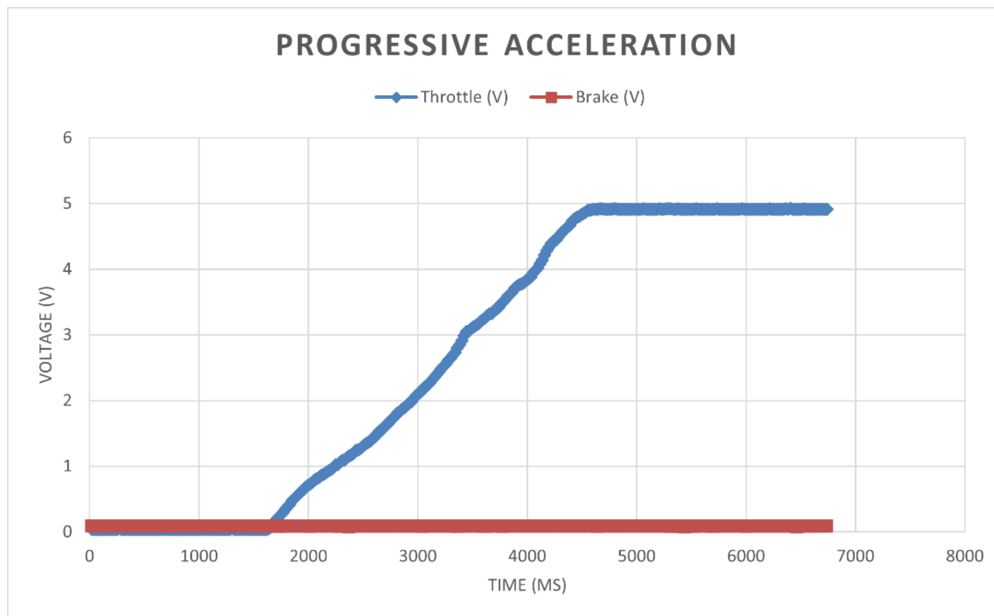


FIGURE 28. PROGRESSIVE ACCELERATION GRAPH. (IMAGE FROM [OWN ELABORATION])

Totally the opposite to the first graph, a hard braking situation was tested as well. The throttle pedal is freed and when it reaches its lowest point the brakes are slammed to their maximum value, meaning that the vehicle will stop in the shortest distance if no other issue arises. (This situation is represented in Figure 29)

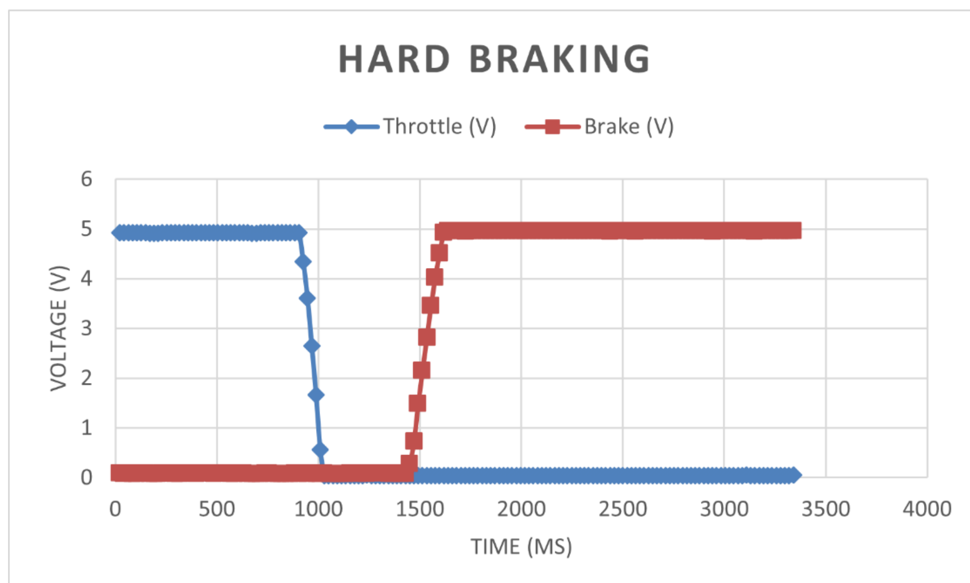


FIGURE 29. HARD BRAKING GRAPH. (IMAGE FROM [OWN ELABORATION])

After these tests, the potential of the project and the system are evaluated. It is important to consider that these throttle-braking graphic analytics can be extended to all the parameters inside the Horizon 20.

5. CONCLUSION

With the goals on one hand and the final results on the other, the overall project is evaluated. The conclusion of the work takes into account two factors, the comparison of results obtained with regards to the objectives set at the very beginning, and possible improvements that have been appearing during the process and couldn’t be included in this first design.

5.1. RESULTS

Starting with the first main goal, the proper motor controller set up was achieved. The creation of a powertrain model allowed to connect each of the signal desired and the communication with the unit’s software through a PC. No further set up or real testing conditions were possible due to the need for a high voltage source and an extensive engine resolver calibration.

Taking now the data logging as focus, the standards set for this part were satisfied. The application has been proved successful in registering all the data the controller is sending through the serial port. The documents in which the information is saved are common file extensions allowing for easy understanding and post analysis. Moreover, the data logging is performed automatically when the code is run with no extra steps required by the user.

Last, meanwhile the data logging is working, the Graphical User Interface developed is performing at the same time its expected function. After testing, the information was found displaying with no delay. The dashboard incorporated visually appealing and intuitive elements to enhance the driver's understanding and monitoring capabilities. The packaging inside the car follows trendy designs in the motorsport industry and it is a revolution inside the Formula Student competition.

Overall, this project successfully accomplished its objectives by creating a motor controller-based system capable of logging representative data from the car. The implemented Python code demonstrated its ability to effectively capture and structure essential parameters, while the graphical user interface provided real-time visualization to enhance the driver's experience. This system holds great potential for applications in car monitoring and performance optimization, and further advancements could be made by incorporating additional features and expanding the range of captured data. (A picture with the final result implemented in the car is shown in Figure 30)

FIGURE 30. DASHBOARD FINAL RESULT. (IMAGE FROM [OWN ELABORATION])

The completion of this project not only enhances the understanding of motor control systems but also lays a solid foundation for future research and development in the field of car telemetry and monitoring.

5.2. FUTURE STEPS

With no doubt the system can be expanded and improved in each of its areas of development. The definition of future steps can be used by UTRON Racing as a road map to know the next tasks to be fulfilled. These advancements can englobe points that will definitely make its implementation in the Horizon 20 possible or details that will make it more user-friendly.

From the data logging point of view, the information is well structured and stored, but if we aim to its analysis, still some manual actions are required. The code could be deeply developed so the remarkable information is already evaluated by the system itself in real-time. Therefore, the user would receive an automatically generated report of the key parts of the information.

Moving to the GUI area there are some aspects to be considered as well. In the first place it would be useful to add a warning message when the information is not being displayed in real-time, *i.e.* communication is failing. That means that the program would not be using the last values in the "queue" and there is a certain delay. Also, the panel or the screen itself could include a series of buttons making the display interactive with the driver. This way the driver would be able to choose the information that he wants to see among different windows and manage the start and stop of the program from the cockpit. This step would level the designed dashboard up with the ones used by high performance cars seen in professional competitions.

If we think about the whole system, the steps are much more ambitious yet achievable. Some extra parameters that don't come from the inverter are expected to be measured, such as wheel temperature, battery accumulator temperature and more. These types of sensors are still to be studied and assembled but most likely they will require a new communication method. In the same direction, the dashboard background design can be modified to count on other parameters that can be registered. Moreover, when the accumulator is complete and the resolver from the engine is calibrated, the controller will receive realistic values from the engine.

Last, the full concept of the system can be reimagined. In this project the main communication was centered on the serial network that the motor controller incorporates. This method was used because it didn't require further development and study before the data started to be received, but the controller also works over two CAN channels. The communication part of the system could be transformed to work with this type of network. This change would take advantage of CAN features to make the system more reliable and faster. Moreover, CAN is easily scalable, meaning that extra sensors, actuators, and ECUs can be added any time. In addition, taking into account the way the GUI code is designed, it shouldn't be greatly affected by the transformation, meaning it could be modified and recycled.

These changes can be seen as suggestions or advice on next steps that, due to the final degree project frame time, couldn't be included in the targets. Anyway, they can encourage UTRON Racing members to ask themselves the following question: "How can I do this a bit better?".

6. BIBLIOGRAPHY

[1] Formula Student Germany. (2022). *Formula Student Rules 2023* [Manual]. Wiesbaden, Germany: Formula Student Germany Organization.

[2] Formula Student Germany. (2023). *Home page*. FSG Website. https://www.formulastudent.de/fsg/

[3] Formula Student Spain. (2023). *Home page*. FSS Website. https://www.formulastudent.es/

[4] UTRON Racing Team. (2023). *Electric Vehicle*. UTRON Racing Website. https://mon.uvic.cat/utronracing/electric-vehicle/?lang=en

[5] Garín, M. (2021). *Vehicles Elèctrics: Powertrain* [Academic notes]. UVic -UCC Moodle.

[6] García, O. (2021). *Desenvolupament del Sistema de Tracció i Circuit de Seguretat d'un Formula Student* (Treball de Fi de Grau, Universitat de Vic).

[7] Caleb's Engineering Projects. (2021). *How to Design an Electric Powertrain (FSAE)*. YouTube. https://youtu.be/Hg7MXIgeig4

[8] EMRAX. (2020). *Motor Installation and Maintenance Manual* [Manual]. Slovenija, EU: EMRAX INNOVATIVE E-DRIVES.

[9] JAES Company. (2022). *What is a SYNCHRONOUS MOTOR and how does it work? - Rotating magnetic field - Synchronism speed.* YouTube. https://youtu.be/Tk3lNBSAgEg

[10] EMRAX. (2020). *EMRAX 228* [Datasheet]. Slovenija, EU: EMRAX INNOVATIVE E-DRIVES.

[11] JAES Company. (2021). *What is an INVERTER and how it works - Sine wave - Square wave - 3D animation.* YouTube. https://youtu.be/iCJ_aMH5gUs

[12] CASCADIA MOTION. (2020). *PM100* [Datasheet]. Wilsonville, OR: CASCADIA MOTION.

[13] Garín, M. (2021). *Vehicles Elèctrics: Convertidors* [Academic notes]. UVic -UCC Moodle.

[14] García, O. (2022). *Justificación de Motor y Baterías* [Internal Documentation]. Spain, EU: UTRON Racing Team.

[15] Energus Power Solutions. (2021). *Li-ion building block with LG HE2 datasheet* [Datasheet]. Lithuania, EU: Energus Power Solutions.

[16] Formula E. (2022). *Formula E And FIA Reveal All-Electric Gen3 Race Car In Monaco.* FIA. https://www.fiaformulae.com/en/news/2456/formula-e-and-fia-reveal-all-electric-gen3-race-car-in-monaco

[17] The Undercut. (2022). *The Insane TECHNOLOGY Behind The NEW F1 Steering Wheel.* YouTube. https://youtu.be/3CWbfLpe7GQ

[18] CASCADIA MOTION. (2011). *PM100 Series AC Motor Drive User's Manual* [Manual]. Wilsonville, OR: CASCADIA MOTION.

[19] CASCADIA MOTION. (2021). *Cascadia Motion Getting Started Guide* [Manual]. Wilsonville, OR: CASCADIA MOTION.

[20] TE Connectivity. (2014). *AMPSEAL* Automation Plug Connector and Header Assembly* [Manual]. Switzerland, EU: TE Connectivity.

[21] TE Connectivity. (2022). *AMPSEAL 35p* [Datasheet]. Switzerland, EU: TE Connectivity.

[22] TE Connectivity. (2022). *AMPSEAL 23p* [Datasheet]. Switzerland, EU: TE Connectivity.

[23] CASCADIA MOTION. (2021). *Software User Manual* [Manual]. Wilsonville, OR: CASCADIA MOTION.

[24] CASCADIA MOTION. (2022). *Download Diagnostic Data* [Manual]. Wilsonville, OR: CASCADIA MOTION.

[25] GitHub. (2020). *pySerial's documentation.* pySerial. https://pyserial.readthedocs.io/en/latest/index.html

[26] Python Software Foundation. (2023). *csv — CSV File Reading and Writing.* Python Software Foundation. https://docs.python.org/3/library/csv.html

[27] Gazoni, E. & Clark, C. (2023). *openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files.* Clark Consulting & Research. https://openpyxl.readthedocs.io/en/stable/

[28] Mercedes-AMG Petronas Formula Team. (2022). *How Much Data Does an F1 Car Generate?.* YouTube. https://youtu.be/JDkePD2-scY

[29] Formula Daily. (2022). *Ferrari F1 Steering Wheel & Dash EXPLAINED.* YouTube. https://youtu.be/dnOpyyHDPbg

[30] Cerdan, A. (2015). *Disseny i millora de la Dashboard del CAT08e, monoplaça de formula student elèctric de l'equip ETSEIB Motorsport.* (Treball de Fi de Grau, Universitat Politècnica de Catalunya).

[31] Vince Opra. (2022). *Photoshop Tutorial for Beginners 2023 | Everything You NEED to KNOW!.* YouTube. https://youtu.be/gKFZjRV7L_Q

[32] STONTRONICS Ltd. (2021). *Raspberry Pi 3 Model B* [Manual]. United Kingdom, EU: RS Components.

[33] Raspberry Pi Ltd. (2023). *Raspberry Pi Documentation.* Raspberry Pi. https://www.raspberrypi.com/documentation/

[34] ELECROW. (2023). *5inch HDMI Display User Manual* [Manual]. Shenzhen, China: ELECROW.

[35] GTK Team. (2023). *Setting up GTK for Windows.* GTK. https://www.gtk.org/docs/installations/windows

[36] Gi-docgen. (2023). *GTK – 3.0: The GTK toolkit.* GTK Team. https://docs.gtk.org/gtk3/

[37] Cairo Graphics. (2023). *Cairo Graphics - Documentation.* Cairo Graphics. https://www.cairographics.org/documentation/

[38] Python Software Foundation. (2023). *threading — Thread-based parallelism.* Python Software Foundation. https://docs.python.org/3/library/threading.html

[39] LYH Studio. (2023). *Learn Adobe InDesign in 11 MINUTES! | Formatting, Tools, Layout, Text Etc.* YouTube. https://youtu.be/wF_fu1wcT0Y

[40] Adobe Inc.. (2016). *ADOBE PREMIERE PRO* [Manual]. San José, CA: Adobe Inc.

7. ANNEXES

ANNEX A

Project planning

Calendar week	Project week	Stages	Tasks
Week 08	Week 01	Project Bases	- Project objectives definition - Needed material's list - Project planning - UTRON's powertrain study
Week 09	Week 02	Project Preparation	- Material purchasement - Wires and adaptors preparation
Week 10	Week 03	Model preparation	- Distribution of main components in the model base - Set up of controller, engine, wires and actuators - Connection and performance check
Week 11	Week 04	Controller software study	- Controller software download - Controller software testing and study of features
Week 12	Week 05	SCI code	- Research about serial port and its functioning - Creation of a Serial Communication Interface using Python - Revision of the format and data type
Week 13	Week 06	Data logging code I	- Research about python libraries that log data in different formats - Definition of parameters that will be registered - Organization of the information through coding
Week 14	Week 07	Data logging code II	- Data translation through coding - Implementation of the libraries - Revision of the documents and formats that the code generates
Week 15	Week 08	Dashboard manufacturing	- Adquisition of hardware components and proper preparation - Measuring of parts and design of housing in CAD software - Housing printing and cockpit plate manufacturing
Week 16	Week 09	GUI programming I	- Study of basic programming functions that create a dashboard - Interface design thinking about the information that is going to be shown - Drawing programming tests
Week 17	Week 10	GUI programming II	- Dashboard development - Dashboard set up commands implementation - Revision of the code
Week 18	Week 11	Project testing	- Testing and polishing of all the different programs over the powertrain model - Memory structure definition
Week 19	Week 12	Memory preparation I	- Introduction writing - State of the Art writing - Beginning of project development explanation
Week 20	Week 13	Memory preparation II	- End of project developement explanation - Results and conclusions writing
Week 21	Week 14	Memory preparation III	- Definition of bibliography and annexes - Abstract writing - Brainstorming about attached documents
Week 22	Week 15	Memory delivery	- Preparation of attached documents (Video, images & summary) - Revision of attached documents
Week 23	Week 16	Presentation preparation	- Presentation structure definition - Presentation file creation
Week 24	Week 17	Project presentation	- Presentation rehearsal

ANNEX B

AMPSEAL J1-35p connections

Main color	Secondary color	Controller pin	Output
		J1-p1	Accel Pedal Power
		J1-p2	Accel Pedal GND
		J1-p3	Analog Input 4 0-5VFS
		J1-p4	1000 Ohm RTD Input
		J1-p5	100 Ohm RTD Input
		J1-p6	100 Ohm RTD Input
		J1-p7	Serial Boot Loader enable
		J1-p8	Reverse Enable Switch
		J1-p9	Ignition Input (if used)
		J1-p10	DO NOT CONNECT
		J1-p11	CAN Channel A Low
		J1-p12	RS-232 Transmit
		J1-p13	Accel Pedal wiper
		J1-p14	Spare 5V transducer power
		J1-p15	Analog Ground
		J1-p16	1000 Ohm RTD Input
		J1-p17	Analog Ground
		J1-p18	DO NOT CONNECT
		J1-p19	Analog Ground
		J1-p20	Brake Switch
		J1-p21	Start Input (if used)
		J1-p22	Ground
		J1-p23	CAN Channel B Hi
		J1-p24	Motor Temperature Sensor
		J1-p25	Brake Pedal
		J1-p26	Spare 5V transducer power
		J1-p27	1000 Ohm RTD Input
		J1-p28	Spare 5V transducer power
		J1-p29	DO NOT CONNECT
		J1-p30	Forward Enable Switch
		J1-p31	REGEN Disable Input (if used)
		J1-p32	DO NOT CONNECT
		J1-p33	CAN Channel A Hi
		J1-p34	CAN Channel B Low
		J1-p35	RS-232 Receive

AMPSEAL J2-23p connections

Main color	Secondary color	Controller pin	Output
		J2-p1	Encoder Power
		J2-p2	Encoder Channel Z input (Index)
		J2-p3	Resolver excitation return
		J2-p4	Resolver Cosine winding +
		J2-p5	DO NOT CONNECT
		J2-p6	Chassis GND
		J2-p7	Main Relay Drive
		J2-p8	12V Ignition Power Input
		J2-p9	Encoder Channel A input (Used with Induction Motors)
		J2-p10	Encoder GND
		J2-p11	Resolver Sine winding +
		J2-p12	Resolver Cosine winding -
		J2-p13	DO NOT CONNECT
		J2-p14	Chassis GND
		J2-p15	OK Indicator Drive / 12V Power Relay Drive
		J2-p16	Encoder Channel B input
		J2-p17	Resolver excitation output (Used with PM Motors)
		J2-p18	Resolver Sine winding -
		J2-p19	Resolver Shield GND
		J2-p20	DO NOT CONNECT
		J2-p21	Pre-Charge Contactor Drive
		J2-p22	Fault Indicator Drive
		J2-p23	12V Ignition Power Input

ANNEX C

Serial Comunication Code

```
import serial
import sys

#Class creation for data manipulation
class Record():

    def __init__(self, data):
        self.data = data

    @classmethod
    def from_bytes(cls, buffer):
        buffer = buffer.strip() #trim of the values that are not for data
purposes
        bdat = buffer.split(b" ") #definition of data organization

        data = [int(x,base=16) for x in bdat] #translation of hexadecimal to
decimal

        return cls(data)

    def __str__(self):  #5 spaces between data values

        s = ''
        for dat in self.data:
            s += f"{dat:5}, "
        s += '*'
        return s

#Serial port configuration
ser = serial.Serial(port="COM7", #serial port performance parameters
                    baudrate=57600,
                    bytesize=8,
                    parity='N',
                    stopbits=1,
```

```
                    timeout=1,
                    xonxoff=0,
                    rtscts=0)

# Empty buffer and wait for new line.
ser.read_all()
ser.readline()

# Start SCI mode.
ser.write(b"+\n\r")

# Discard first line
ser.readline()

#Loop creation for reading and printing lines
while True:

    if sys.stdin.read(1):
        break
    line = ser.readline(); #read line and keep it, don't discard it
    if not line:
        # start SCI mode.
        ser.write(b"+\n\r")

        # Discard first line
        ser.readline()

        continue

    record = Record.from_bytes(line)
    print(record)

# stop SCI mode. (toggle)
ser.write(b"+\n\r")

#safely close serial port
ser.close()
```

ANNEX D

Data Acquisition Parameters

1	The low word of the Power On Timer (increments every 3ms)
2	Filtered Accel-pot input voltage (V) times 100
3	Motor Torque feedback (Nm) times 10
4	Vehicle Torque Command (Nm) times 10
5	DC Voltage (V) times 10
6	DC Current (V) times 10
7	Motor Speed (rpm)
8	Flux Weakening Regulator Output (Apk) times 10
9	Motor Voltage Magnitude (Vpk) times 10
10	IQ Command (Apk) times 10
11	IQ Feedback (Apk) times 10
12	ID Command (Apk) times 10
13	ID Feedback (Apk) times 10
14	Modulation times 10000
15	Module A Temperature (°C) times 10
16	Motor Temperature (°C) times 10
17	Run Fault Low Word
18	Run Fault High Word
19	Torque Shudder (Nm) times 10
20	Filtered Brake pot (V) times 100

ANNEX E

Motor speed to vehicle speed conversion

$$Z = gear\ teeth$$

$$Z_{pinion} = 15\ teeth$$

$$Z_{crown} = 43\ teeth$$

$$r_{wheel} = 0'2625\ m$$

$$\frac{\omega_{crown}}{\omega_{pinion}} = \frac{Z_{pinion}}{Z_{crown}}$$

$$\omega_{crown} = \frac{Z_{pinion}}{Z_{crown}} \cdot \omega_{pinion}$$

$$V_{HORIZON} = \omega_{crown} \cdot 2\pi \cdot\ r_{wheel}$$

$$V_{HORIZON} = \omega_{pinion} \cdot \frac{Z_{pinion}}{Z_{crown}} \cdot 2\pi \cdot\ r_{wheel}$$

$$V_{HORIZON} = \omega_{pinion} \cdot \frac{15}{43} \cdot 2\pi \cdot\ 0'2625\ \ (m/min)$$

$$V_{HORIZON} = \omega_{pinion} \cdot \frac{15}{43} \cdot 2\pi \cdot\ 0'0002625 \cdot 60\ (km/h)$$

$$V_{HORIZON} = \omega_{pinion} \cdot 0'0345\ \left(\frac{km}{h}\right)$$

$$Conversion\ factor = 0'0345 = \frac{1}{28'99}$$

It means any angular speed value coming from the electric engine must be multiplied by 0’0345 or divided by 28’99 to obtain the car linear speed in kilometers per hour.

ANNEX F

Data Logging Code

```
from datetime import datetime

from openpyxl import Workbook

from openpyxl.styles import Font

import keyboard

import csv


#Data string simulation for test purposes

record_string= '2673, 0492, 0342, 0856, 3456, 2451, 9600, 8734, 3345, 7654,
2673, 0492, 0342, 0856, 3456, 2451, 9600, 8734, 3345, 7654'

record = str(record_string)

recordlist = record.split(",")


while True:

    with open('CSV file.txt', 'a', newline='') as file:

        writer = csv.writer(file, quoting=csv.QUOTE_ALL,delimiter=';')

        writer.writerow(recordlist)

        print(recordlist)

        #x+=1

    if keyboard.is_pressed('esc'):

        print("The ESC button was pressed. The files are being saved.")

        break


ft = Font(bold=True)

wb= Workbook()
```

```
#Version sheet configuration
wsversion=wb.active
wsversion.title='Version'
wsversion['A1'].font=ft
wsversion['A1']="Horizon 20 Data Adquisition File"
wsversion['A2'].font=ft
wsversion['A2']='Date'
wsversion['A3'].font=ft
wsversion['A3']='Time'
wsversion['B2']=datetime.today().strftime('%Y-%m-%d')
wsversion['B3']=datetime.now().strftime("%H:%M:%S")

#Data sheets creation
wsthrottle = wb.create_sheet("Throttle")
wsbrake = wb.create_sheet("Brake")
wsmotortorque = wb.create_sheet("Motor Torque")
wsvehicletorque = wb.create_sheet("Vehicle Torque")
wsdcvoltage = wb.create_sheet("DC Voltage")
wsdccurrent = wb.create_sheet("DC Current")
wsvehiclespeed = wb.create_sheet("Vehicle Speed")
wsmotorspeed = wb.create_sheet("Motor Speed")
wsmotorvoltage = wb.create_sheet("Motor Voltage")
wscontrollertemp = wb.create_sheet("Controller Temperature")
wsmotortemp= wb.create_sheet("Motor Temperature")

#Tables creation
wsthrottle['A1'].font=ft
wsthrottle['A1']='Time (ms)'
```

```
wsthrottle['B1'].font=ft
wsthrottle['B1']='Throttle (V)'

wsbrake['A1'].font=ft
wsbrake['A1']='Time (ms)'
wsbrake['B1'].font=ft
wsbrake['B1']='Brake (V)'

wsmotortorque['A1'].font=ft
wsmotortorque['A1']='Time (ms)'
wsmotortorque['B1'].font=ft
wsmotortorque['B1']='Motor Torque (Nm)'

wsvehicletorque['A1'].font=ft
wsvehicletorque['A1']='Time (ms)'
wsvehicletorque['B1'].font=ft
wsvehicletorque['B1']='Vehicle Torque (Nm)'

wsdcvoltage['A1'].font=ft
wsdcvoltage['A1']='Time (ms)'
wsdcvoltage['B1'].font=ft
wsdcvoltage['B1']='DC Voltage (V)'

wsdccurrent['A1'].font=ft
wsdccurrent['A1']='Time (ms)'
wsdccurrent['B1'].font=ft
wsdccurrent['B1']='DC Current (A)'
```

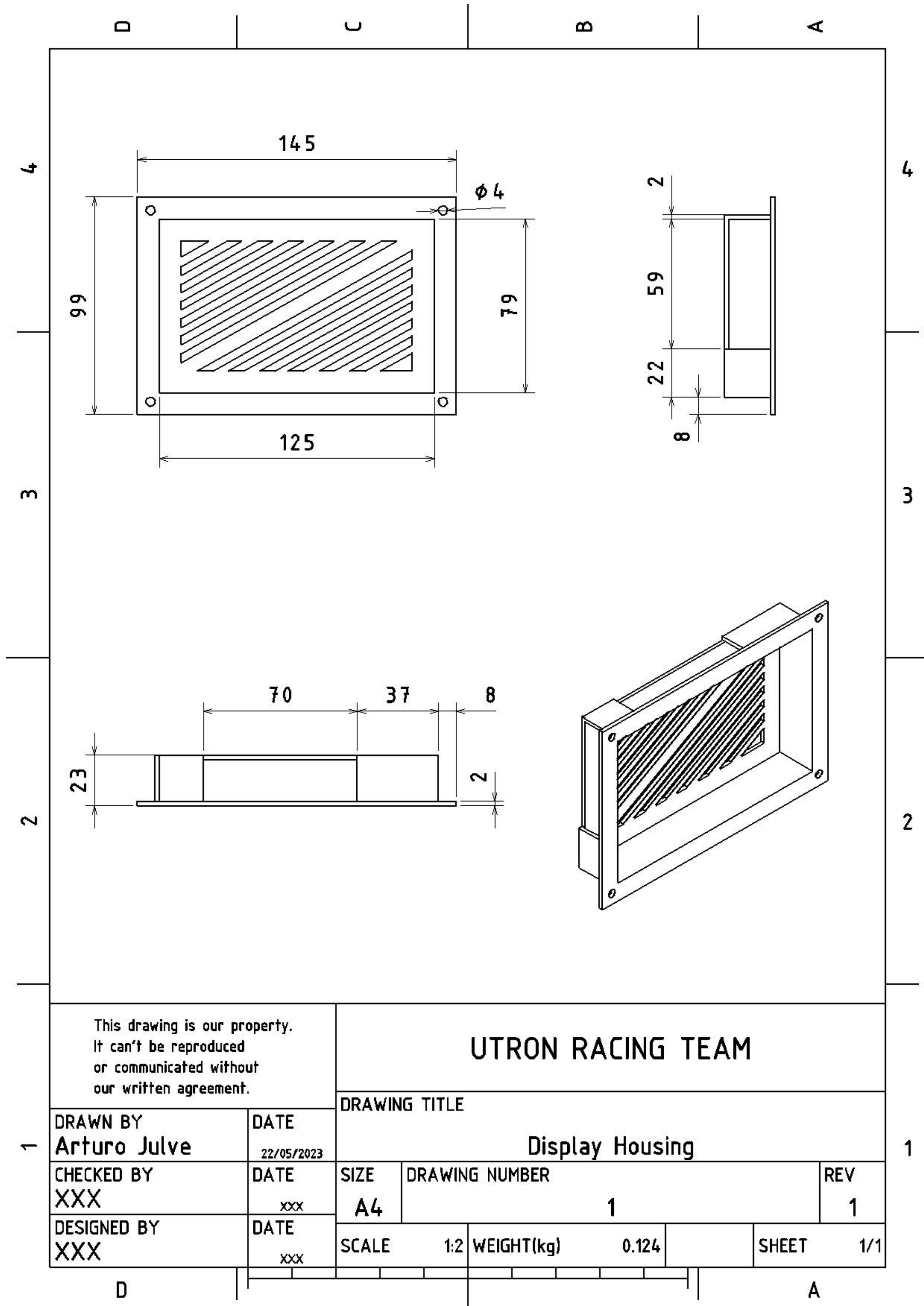
```
wsmotorvoltage['A1'].font=ft
wsmotorvoltage['A1']='Time (ms)'
wsmotorvoltage['B1'].font=ft
wsmotorvoltage['B1']='Motor Voltage (Vpk)'

wsvehiclespeed['A1'].font=ft
wsvehiclespeed['A1']='Time (ms)'
wsvehiclespeed['B1'].font=ft
wsvehiclespeed['B1']='Vehicle Speed (km/h)'

wsmotorspeed['A1'].font=ft
wsmotorspeed['A1']='Time (ms)'
wsmotorspeed['B1'].font=ft
wsmotorspeed['B1']='Motor Speed (rpm)'

wscontrollertemp['A1'].font=ft
wscontrollertemp['A1']='Time (ms)'
wscontrollertemp['B1'].font=ft
wscontrollertemp['B1']='Controller Temperature (ºC)'

wsmotortemp['A1'].font=ft
wsmotortemp['A1']='Time (ms)'
wsmotortemp['B1'].font=ft
wsmotortemp['B1']='Motor Temperature (ºC)'

#CSV reading and data segmentation
with open('CSV file.txt', 'r', newline='') as file:
    reader= csv.reader(file, quoting=csv.QUOTE_ALL,delimiter=';')
```

```
for line in reader:
    data_list=line
    time=int(data_list[0])
    throttle=int(data_list[1])
    motor_torque=int(data_list[2])
    vehicle_torque=int(data_list[3])
    dc_voltage=int(data_list[4])
    dc_current=int(data_list[5])
    motor_speed=int(data_list[6])
    motor_voltage=int(data_list[8])
    controller_temp=int(data_list[14])
    motor_temp=int(data_list[15])
    brake=int(data_list[19])

    #Data distribution and conversion
    throttlelist=[time,throttle/100]
    brakelist=[time,brake/100]
    motortorquelist=[time, motor_torque/10]
    vehicletorquelist=[time, vehicle_torque/10]
    dcvoltagelist=[time, dc_voltage/10]
    dccurrentlist=[time, dc_current/10]
    vehiclespeedlist=[time, motor_speed/28.99]
    motorspeedlist=[time, motor_speed]
    motorvoltagelist=[time, motor_voltage/10]
    controllertemplist=[time, controller_temp/10]
    motortemplist=[time, motor_temp/10]

    #Data storage
```

```
        wsthrottle.append(throttlelist)

        wsbrake.append(brakelist)

        wsmotortorque.append(motortorquelist)

        wsvehicletorque.append(vehicletorquelist)

        wsdcvoltage.append(dcvoltagelist)

        wsdccurrent.append(dccurrentlist)

        wsvehiclespeed.append(vehiclespeedlist)

        wsmotorspeed.append(motorspeedlist)

        wsmotorvoltage.append(motorvoltagelist)

        wscontrollertemp.append(controllertemplist)

        wsmotortemp.append(motortemplist)


wb.save('Horizon                                                  20
Data_'+str(datetime.today().strftime('%Y_%m_%d_'))+str(datetime.now().strftim
e("%H_%M"))+'.xlsx')

print('Program finished')
```

ANNEX G

Display housing general dimensions



ANNEX H

Graphical User Interface Code

```
import gi

gi.require_version("Gtk", "3.0")

gi.require_version('PangoCairo', '1.0')

from gi.repository import Gtk,GLib,Gdk,GdkPixbuf, Pango, cairo




class UTRONwindow(Gtk.Window):


    def __init__(self,q):


        super().__init__()

        self.set_default_size(800,480)

        self.connect("destroy", self.on_destroy)


        # Set an icon for the window.

        img = Gtk.Image.new_from_file("Icon UTRON.png")

        self.set_icon(img.get_pixbuf())

        self.q = q


    def do_key_press_event(self,event):

        # Close the app pressing scape.

        # This is handy as there is no button to close the app in fullscreen
mode.

        if(event.keyval == Gdk.KEY_Escape):

            self.close()
```

```
    def on_destroy(self,*args):

        print("on_destroy")

        Gtk.main_quit()


    def hide_cursor(self):

        """

        Makes the cursor invisible.

        Inportant! This can be only called after the window is

        physically displayed (i.e. after show() is called).

        """

        win = self.get_window() # get Gdk window.

        disp = win.get_display()

        cursor = Gdk.Cursor.new_for_display(disp,Gdk.CursorType.BLANK_CURSOR)

        win.set_cursor(cursor)


    def run(self):

        #self.maximize()

        #self.fullscreen()

        self.show_all()

        self.hide_cursor()


        # Makes sure it stays above any other window.

        self.set_keep_above(True)

        Gtk.main()


class UTRONDrawingArea(Gtk.DrawingArea):
```

```
    def __init__(self):
        super().__init__()

        # Load the image file
        pixbuf                                                              =
GdkPixbuf.Pixbuf.new_from_file("New_Dashboard_Template_Dark.png")

        # Create a Gtk.Image from the Pixbuf
        self.image = Gtk.Image.new_from_pixbuf(pixbuf)

        # Connect the 'draw' signal to the drawing method
        self.connect("draw", self.on_draw)

    def on_draw(self, widget, cr):
        # Get the size of the DrawingArea
        width = self.get_allocated_width()
        height = self.get_allocated_height()

        # Scale the image to fit the DrawingArea
        scaled_pixbuf  =  self.image.get_pixbuf().scale_simple(width,  height,
GdkPixbuf.InterpType.BILINEAR)

        # Draw the scaled image onto the Cairo context
        Gdk.cairo_set_source_pixbuf(cr, scaled_pixbuf, 0, 0)
        cr.paint()

        # Set the color of the rectangle to red
        cr.set_source_rgb(1.0, 0.0, 0.0)
```

```
# Conversion of the real value to the pixel value
brake = 10
brakevalue= brake/100 # maximum brake value is 4.93
brakedrawing=brakevalue*83.37 # multiply to scale the dimensions
# Draw the rectangle onto the Cairo context
cr.rectangle(12, 434-brakedrawing, 94, brakedrawing)
# Fill the rectangle
cr.fill()

# Throttle drawing
cr.set_source_rgb(0.0, 1.0, 1.0)
throttle = 350
throttlevalue= throttle/100 # maximum throttle value is 4.93
throttledrawing=throttlevalue*83.37 # multiply to scale the dimensions
cr.rectangle(130, 434-throttledrawing, 94, throttledrawing)
cr.fill()

# RPM writing
# Set the font for the text
cr.set_font_size(48)
# Set the color for the text
cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color
# Draw the text
motor_speed = 3333 # max RPM value is 6500
RPMstring=str(motor_speed)
(x, y, width, height, dx, dy) = cr.text_extents(RPMstring)
cr.move_to(265,257)
cr.show_text(RPMstring)
```

```
        # Motor torque writing

        cr.set_font_size(48)

        cr.set_source_rgb(1.0, 1.0, 1.0)

        motor_torque = 4900

        motortorquevalue= motor_torque/10

        motortorquedrawing = str(motortorquevalue)

        (x, y, width, height, dx, dy) = cr.text_extents(motortorquedrawing)

        cr.move_to(395,257)

        cr.show_text(motortorquedrawing)


        # Vehicle speed writing

        cr.set_font_size(100)

        cr.set_source_rgb(1.0, 1.0, 1.0)

        speedfloat = float(motor_speed/28.97)

        speedint = int(speedfloat)

        speed = str(speedint)

        (x, y, width, height, dx, dy) = cr.text_extents(speed)

        cr.move_to(335,190)

        cr.show_text(speed)


        #DC current drawing

        cr.set_source_rgb(1.0, 0.0, 1.0)

        dc_current= 5000

        dccurrentvalue= int(dc_current/10) #maximum dccurrent value is 500

        dccurrentdrawing=dccurrentvalue*0.384   #   multiply   to   scale   the
dimensions

        cr.rectangle(263, 414, dccurrentdrawing, 20)
```

```
        cr.fill()


        # DC current writing

        cr.set_font_size(24)

        cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color

        (x, y, width, height, dx, dy) = cr.text_extents(str(dccurrentvalue))

        cr.move_to(460,432)

        cr.show_text(str(dccurrentvalue) + ' A')


        #DC voltage drawing

        cr.set_source_rgb(1.0, 0.0, 1.0)

        dc_voltage= 3000

        dcvoltagevalue= int(dc_voltage/10) #maximum dcvoltage value is 700

        dcvoltagedrawing=dcvoltagevalue*0.27   #   multiply   to   scale   the
dimensions

        cr.rectangle(263, 353, dcvoltagedrawing, 20)

        cr.fill()


        # DC voltage writing

        cr.set_font_size(24)

        cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color

        (x, y, width, height, dx, dy) = cr.text_extents(str(dcvoltagevalue))

        cr.move_to(460,371)

        cr.show_text(str(dcvoltagevalue) + ' V')


        # Motor voltage drawing

        cr.set_source_rgb(1.0, 0.0, 1.0)

        motor_voltage=5000
```

```
        motorvoltagevalue= int(motor_voltage/10) #maximum motor voltage value
is 700

        motorvoltagedrawing=motorvoltagevalue*0.27  #  multiply  to  scale  the
dimensions

        cr.rectangle(263, 291, motorvoltagedrawing, 20)

        cr.fill()


        # Motor voltage writing

        cr.set_font_size(24)

        cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color

        (x, y, width, height, dx, dy) = cr.text_extents(str(motorvoltagevalue))

        cr.move_to(460,310)

        cr.show_text(str(motorvoltagevalue) + ' V')


        # Front left tyre temperature drawing

        fltyretemperature= 70

        if fltyretemperature<50:

            cr.set_source_rgb(0.0, 0.0, 1.0)

        elif 50<=fltyretemperature<=70:

            cr.set_source_rgb(0.0, 1.0, 0.0)

        elif 70<fltyretemperature<=80:

            cr.set_source_rgb(1.0, 0.647, 0.0)

        elif fltyretemperature>80:

            cr.set_source_rgb(1.0, 0.0, 0.0)

        cr.rectangle(542, 57, 67, 99)

        cr.fill()


        # Front right tyre temperature drawing
```

```
frtyretemperature= 70
if frtyretemperature<50:
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=frtyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 70<frtyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif frtyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(719, 57, 67, 99)
cr.fill()

# Rear right tyre temperature drawing
rrtyretemperature= 70
if rrtyretemperature<50:
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=rrtyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 70<rrtyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif rrtyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(719, 291, 67, 100)
cr.fill()

# Rear left tyre temperature drawing
rltyretemperature= 70
if rltyretemperature<50:
```

```
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=rltyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 70<rltyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif rltyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(542, 291, 67, 100)
cr.fill()

# Battery pack temperature drawing
batterytemperature= 55
if batterytemperature<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=batterytemperature<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
elif batterytemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(631, 219, 67, 58)
cr.fill()

# Controller temperature drawing
controller_temp = 700
controllertemperature= controller_temp/10
if controllertemperature<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=controllertemperature<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
```

```
elif controllertemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(631, 290, 67, 22)
cr.fill()

# Engine temperature drawing
motor_temp = 810
enginetemperature= motor_temp/10
if enginetemperature<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=enginetemperature<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
elif enginetemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(651, 326, 29, 54)
cr.fill()

# Warning messages writing
cr.set_font_size(15)
cr.set_source_rgb(1.0, 0.0, 0.0)  # set RGB values for red color
if motorvoltagevalue>600:
    cr.rectangle(263, 40, 248, 64)
    cr.fill()
    cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color
    problem = "High engine voltage"
    action = "!STOP THE CAR!"
    (x, y, width, height, dx, dy) = cr.text_extents(problem)
    cr.move_to(325,65)
```

```
    cr.show_text(problem)
    (x, y, width, height, dx, dy) = cr.text_extents(action)
    cr.move_to(328,90)
    cr.show_text(action)

elif dcvoltagevalue>400:
    cr.rectangle(263, 40, 248, 64)
    cr.fill()
    cr.set_source_rgb(1.0, 1.0, 1.0)
    problem = "High battery voltage"
    action = "!STOP THE CAR!"
    (x, y, width, height, dx, dy) = cr.text_extents(problem)
    cr.move_to(325,65)
    cr.show_text(problem)
    (x, y, width, height, dx, dy) = cr.text_extents(action)
    cr.move_to(328,90)
    cr.show_text(action)

elif dccurrentvalue>400:
    cr.rectangle(263, 40, 248, 64)
    cr.fill()
    cr.set_source_rgb(1.0, 1.0, 1.0)
    problem = "High battery current"
    action = "!STOP THE CAR!"
    (x, y, width, height, dx, dy) = cr.text_extents(problem)
    cr.move_to(325,65)
    cr.show_text(problem)
    (x, y, width, height, dx, dy) = cr.text_extents(action)
```

```
    cr.move_to(328,90)

    cr.show_text(action)


elif brakevalue<2:

    cr.rectangle(263, 40, 248, 64)

    cr.fill()

    cr.set_source_rgb(1.0, 1.0, 1.0)

    problem = "No brakes detected"

    action = "!STOP CAREFULLY!"

    (x, y, width, height, dx, dy) = cr.text_extents(problem)

    cr.move_to(325,65)

    cr.show_text(problem)

    (x, y, width, height, dx, dy) = cr.text_extents(action)

    cr.move_to(328,90)

    cr.show_text(action)


elif batterytemperature>80:

    cr.rectangle(263, 40, 248, 64)

    cr.fill()

    cr.set_source_rgb(1.0, 1.0, 1.0)

    problem = "Battery overheating"

    action = "!COOL CAR DOWN!"

    (x, y, width, height, dx, dy) = cr.text_extents(problem)

    cr.move_to(325,65)

    cr.show_text(problem)

    (x, y, width, height, dx, dy) = cr.text_extents(action)

    cr.move_to(328,90)

    cr.show_text(action)
```

```
elif controllertemperature>80:

    cr.rectangle(263, 40, 248, 64)

    cr.fill()

    cr.set_source_rgb(1.0, 1.0, 1.0)

    problem = "Controller overheating"

    action = "!COOL DOWN CAR!"

    (x, y, width, height, dx, dy) = cr.text_extents(problem)

    cr.move_to(325,65)

    cr.show_text(problem)

    (x, y, width, height, dx, dy) = cr.text_extents(action)

    cr.move_to(328,90)

    cr.show_text(action)


elif enginetemperature>80:

    cr.rectangle(263, 40, 248, 64)

    cr.fill()

    cr.set_source_rgb(1.0, 1.0, 1.0)

    problem = "Engine overheating"

    action = "!COOL CAR DOWN!"

    (x, y, width, height, dx, dy) = cr.text_extents(problem)

    cr.move_to(325,65)

    cr.show_text(problem)

    (x, y, width, height, dx, dy) = cr.text_extents(action)

    cr.move_to(328,90)

    cr.show_text(action)
```

```
        elif fltyretemperature>70 or frtyretemperature>70 or rltyretemperature>70 or rrtyretemperature>70:

            cr.rectangle(263, 40, 248, 64)

            cr.fill()

            cr.set_source_rgb(1.0, 1.0, 1.0)

            problem = "Tyre(s) overheating"

            action = "!TOO AGRESSIVE!"

            (x, y, width, height, dx, dy) = cr.text_extents(problem)

            cr.move_to(325,65)

            cr.show_text(problem)

            (x, y, width, height, dx, dy) = cr.text_extents(action)

            cr.move_to(328,90)

            cr.show_text(action)


# Create and configure dashboard.

db = UTRONwindow()


# Create a new DrawingArea

drawing_area = UTRONDrawingArea()


# Add the DrawingArea to the window

db.add(drawing_area)


# Start the application (main loop).

db.run()
```

ANNEX I

UTRON_Excel widget

```
from datetime import datetime
from openpyxl import Workbook
from openpyxl.styles import Font
import csv



class UtronExcel():

    def translation(self):
        ft = Font(bold=True)
        wb= Workbook()
        #Version sheet configuration
        wsversion=wb.active
        wsversion.title='Version'
        wsversion['A1'].font=ft
        wsversion['A1']="Horizon 20 Data Adquisition File"
        wsversion['A2'].font=ft
        wsversion['A2']='Date'
        wsversion['A3'].font=ft
        wsversion['A3']='Time'
        wsversion['B2']=datetime.today().strftime('%Y-%m-%d')
        wsversion['B3']=datetime.now().strftime("%H:%M:%S")

        #Data sheets creation
        wsthrottle = wb.create_sheet("Throttle")
        wsbrake = wb.create_sheet("Brake")
        wsmotortorque = wb.create_sheet("Motor Torque")
        wsvehicletorque = wb.create_sheet("Vehicle Torque")
        wsdcvoltage = wb.create_sheet("DC Voltage")
        wsdccurrent = wb.create_sheet("DC Current")
        wsvehiclespeed = wb.create_sheet("Vehicle Speed")
        wsmotorspeed = wb.create_sheet("Motor Speed")
        wsmotorvoltage = wb.create_sheet("Motor Voltage")
```

```
wscontrollertemp = wb.create_sheet("Controller Temperature")
wsmotortemp= wb.create_sheet("Motor Temperature")

#Tables creation
wsthrottle['A1'].font=ft
wsthrottle['A1']='Time (ms)'
wsthrottle['B1'].font=ft
wsthrottle['B1']='Throttle (V)'

wsbrake['A1'].font=ft
wsbrake['A1']='Time (ms)'
wsbrake['B1'].font=ft
wsbrake['B1']='Brake (V)'

wsmotortorque['A1'].font=ft
wsmotortorque['A1']='Time (ms)'
wsmotortorque['B1'].font=ft
wsmotortorque['B1']='Motor Torque (Nm)'

wsvehicletorque['A1'].font=ft
wsvehicletorque['A1']='Time (ms)'
wsvehicletorque['B1'].font=ft
wsvehicletorque['B1']='Vehicle Torque (Nm)'

wsdcvoltage['A1'].font=ft
wsdcvoltage['A1']='Time (ms)'
wsdcvoltage['B1'].font=ft
wsdcvoltage['B1']='DC Voltage (V)'

wsdccurrent['A1'].font=ft
wsdccurrent['A1']='Time (ms)'
wsdccurrent['B1'].font=ft
wsdccurrent['B1']='DC Current (A)'

wsmotorvoltage['A1'].font=ft
wsmotorvoltage['A1']='Time (ms)'
wsmotorvoltage['B1'].font=ft
```

```
wsmotorvoltage['B1']='Motor Voltage (Vpk)'

wsvehiclespeed['A1'].font=ft
wsvehiclespeed['A1']='Time (ms)'
wsvehiclespeed['B1'].font=ft
wsvehiclespeed['B1']='Vehicle Speed (km/h)'

wsmotorspeed['A1'].font=ft
wsmotorspeed['A1']='Time (ms)'
wsmotorspeed['B1'].font=ft
wsmotorspeed['B1']='Motor Speed (rpm)'

wscontrollertemp['A1'].font=ft
wscontrollertemp['A1']='Time (ms)'
wscontrollertemp['B1'].font=ft
wscontrollertemp['B1']='Controller Temperature (ºC)'

wsmotortemp['A1'].font=ft
wsmotortemp['A1']='Time (ms)'
wsmotortemp['B1'].font=ft
wsmotortemp['B1']='Motor Temperature (ºC)'

with open('Horizon 20 Data in CSV.txt', 'r', newline='') as file:
    reader= csv.reader(file, quoting=csv.QUOTE_ALL,delimiter=',')
    for line in reader:
        data_list=line
        time=int(data_list[0])
        throttle=int(data_list[1])
        motor_torque=int(data_list[2])
        vehicle_torque=int(data_list[3])
        dc_voltage=int(data_list[4])
        dc_current=int(data_list[5])
        motor_speed=int(data_list[6])
        motor_voltage=int(data_list[8])
        controller_temp=int(data_list[14])
        motor_temp=int(data_list[15])
        brake=int(data_list[19])
```

```
                #data distribution and conversion
                throttlelist=[time,throttle/100]
                brakelist=[time,brake/100]
                motortorquelist=[time, motor_torque/10]
                vehicletorquelist=[time, vehicle_torque/10]
                dcvoltagelist=[time, dc_voltage/10]
                dccurrentlist=[time, dc_current/10]
                vehiclespeedlist=[time, motor_speed/28.99]
                motorspeedlist=[time, motor_speed]
                motorvoltagelist=[time, motor_voltage/10]
                controllertemplist=[time, controller_temp/10]
                motortemplist=[time, motor_temp/10]

                #Data storage
                wsthrottle.append(throttlelist)
                wsbrake.append(brakelist)
                wsmotortorque.append(motortorquelist)
                wsvehicletorque.append(vehicletorquelist)
                wsdcvoltage.append(dcvoltagelist)
                wsdccurrent.append(dccurrentlist)
                wsvehiclespeed.append(vehiclespeedlist)
                wsmotorspeed.append(motorspeedlist)
                wsmotorvoltage.append(motorvoltagelist)
                wscontrollertemp.append(controllertemplist)
                wsmotortemp.append(motortemplist)

        #Save file
        wb.save('Horizon 20
Data_'+str(datetime.today().strftime('%Y_%m_%d_'))+str(datetime.now().strftim
e("%H_%M"))+'.xlsx')


# Condition to run code only if it is run in a main section
if __name__ == "__main__":
    excel= UtronExcel()
    excel.translation()
    print("Translation finished")
```

ANNEX J

UTRON_Logger widget

```
import sys
from datetime import datetime
from threading import Thread
import time
from queue import Queue, Full
import signal
import serial

# Class creation for data manipulation
class Record():

    def __init__(self, data):
        self.data = data

    @classmethod
    def from_bytes(cls, buffer):
        buffer = buffer.strip() #trim of the values that are not for data
purposes
        bdat = buffer.split(b" ") #definition of data organization

        data = [int(x,base=16) for x in bdat] #translation of hexadecimal to
decimal

        return cls(data)

    def __str__(self):  #5 spaces between data values

        s = ''
        for dat in self.data:
            s += f"{dat:5}, "
        return s[:-2]

    # Creation of properties that perform over the read data
    @property
```

```
def throttle(self):
    return self.data[1]/100

@property
def motor_torque(self):
    return self.data[2]/10

@property
def dc_voltage(self):
    return self.data[4]/10

@property
def dc_current(self):
    return self.data[5]/10

@property
def motor_speed(self):
    return self.data[6]

@property
def vehicle_speed(self):
    return self.data[6]/28.99

@property
def motor_voltage(self):
    return self.data[8]/10

@property
def controller_temp(self):
    return self.data[14]/10

@property
def motor_temp(self):
    return self.data[15]/10

@property
def brake(self):
```

```
        return self.data[19]/100



# Class creation to manage the different programming flows, set the serial
port and read the data
class UtronLogger(Thread):


    def __init__(self, queue:Queue = None, *args, **kwargs):
        super().__init__(*args,**kwargs)


        self.stop_flag = False
        self.ser = None
        self.queue = queue


    def serial_open(self):


        # Open serial connection.
        self.ser = serial.Serial(port="/dev/ttyUSB0", #serial port
performance paremeters
                                 baudrate=57600,
                                 bytesize=8,
                                 parity='N',
                                 stopbits=1,
                                 timeout=1,
                                 xonxoff=0,
                                 rtscts=0)


        # Empty buffer and wait for new line.
        self.ser.read_all()
        self.ser.readline()


        # Start SCI mode.
        self.ser.write(b"+\n\r")


    def serial_close(self):


        # Stop SCI mode. (toggle)
        self.ser.write(b"+\n\r")
```

```
        self.ser.close()

    def stop(self):
        # Signal to manage thread
        self.stop_flag = True

    def run(self):

        self.serial_open()

        # Discard first line
        self.ser.readline()



        while not self.stop_flag:

            line = self.ser.readline(); #read line and keep it, don't discard
it
            if not line:
                # Start SCI mode.
                self.ser.write(b"+\n\r")

                # Discard first line
                self.ser.readline()

                continue

            record = Record.from_bytes(line)

            print(record)
            sys.stdout.flush()

            if self.queue is not None:

                msg = (record.throttle,
                       record.brake,
```

```
                        record.dc_voltage,
                        record.dc_current,
                        record.motor_torque,
                        record.motor_voltage,
                        record.motor_temp,
                        record.motor_speed,
                        record.controller_temp,
                        record.vehicle_speed,
                        )
                try:
                    self.queue.put(msg, block=False)
                except Full:
                    pass

            with open('Horizon 20 Data in CSV.txt', 'a', newline='') as file:
                file.write(str(record)+"\n")


        self.serial_close()

# Condition to run code only if it is run in a main section
if __name__ == "__main__":
    logger = UtronLogger()

    logger.start()



    def signal_handler(sig, frame):
        print('You pressed Ctrl+C!')
        logger.stop()



    signal.signal(signal.SIGINT, signal_handler)

    logger.join()

    print("Program finished")
```

ANNEX K

UTRON_Dashboard APP

```
import gi
gi.require_version("Gtk", "3.0")
gi.require_version('PangoCairo', '1.0')
from gi.repository import Gtk,GLib,Gdk,GdkPixbuf, Pango, cairo

# Importation of the self-developed widgets
from UTRON_Logger import UtronLogger
from UTRON_Excel import UtronExcel

class UTRONwindow(Gtk.Window):

    def __init__(self,q):

        super().__init__()
        self.set_default_size(800,480)
        self.connect("destroy", self.on_destroy)

        # Set an icon for the window.
        img = Gtk.Image.new_from_file("Icon UTRON.png")
        self.set_icon(img.get_pixbuf())

        # Create a new DrawingArea
        self.dash = UTRONDrawingArea()

        # Add the DrawingArea to the window
        self.add(self.dash)
        self.q = q

    def do_key_press_event(self,event):
        # Close the app pressing scape.
        # This is handy as there is no button to close the app in fullscreen
mode.
        if(event.keyval == Gdk.KEY_Escape):
            self.close()
```

```
def on_destroy(self,*args):
    print("on_destroy")
    Gtk.main_quit()

def on_idle(self):
    try:
        data = self.q.get(block=True,timeout=20e-3)
        self.dash.throttle = data[0]
        self.dash.motor_torque = data[4]
        self.dash.dc_voltage = data[2]
        self.dash.dc_current = data[3]
        self.dash.motor_speed = data[7]
        self.dash.vehicle_speed = data[9]
        self.dash.motor_voltage = data[5]
        self.dash.controller_temp = data[8]
        self.dash.motor_temp = data[6]
        self.dash.brake = data[1]
        self.dash.queue_draw()

    except queue.Empty:
        pass

    return True



def hide_cursor(self):
    """
    Makes the cursor invisible.
    Inportant! This can be only called after the window is
    physically displayed (i.e. after show() is called).
    """
    win = self.get_window() # get Gdk window.
    disp = win.get_display()
    cursor = Gdk.Cursor.new_for_display(disp,Gdk.CursorType.BLANK_CURSOR)
    win.set_cursor(cursor)
```

```
    def run(self):
        self.maximize()
        self.fullscreen()
        self.show_all()
        self.hide_cursor()

        # Make sure it stays above any other window.
        self.set_keep_above(True)

        GLib.idle_add(self.on_idle)
        Gtk.main()

class UTRONDrawingArea(Gtk.DrawingArea):

    def __init__(self):
        super().__init__()

        # Load the image file
        pixbuf =
GdkPixbuf.Pixbuf.new_from_file("New_Dashboard_Template_Dark.png")

        # Create a Gtk.Image from the Pixbuf
        self.image = Gtk.Image.new_from_pixbuf(pixbuf)

        # Connect the 'draw' signal to the drawing method
        self.connect("draw", self.on_draw)

        # Vehicle data.
        self.throttle = 0
        self.motor_torque = 0
        self.dc_voltage = 0
        self.dc_current = 0
        self.motor_speed = 0
        self.vehicle_speed = 0
        self.motor_voltage = 0
        self.controller_temp = 0
```

```
        self.motor_temp = 0
        self.brake = 0

    def on_draw(self, widget, cr):
        # Get the size of the DrawingArea
        width = self.get_allocated_width()
        height = self.get_allocated_height()

        # Scale the image to fit the DrawingArea
        scaled_pixbuf = self.image.get_pixbuf().scale_simple(width, height,
GdkPixbuf.InterpType.BILINEAR)

        # Draw the scaled image onto the Cairo context
        Gdk.cairo_set_source_pixbuf(cr, scaled_pixbuf, 0, 0)
        cr.paint()

        # Set the color of the rectangle to red
        cr.set_source_rgb(1.0, 0.0, 0.0)
        # Conversion of the real value to the pixel value
        brake = self.brake
        brakedrawing=brake*83.37 # multiply to scale the dimensions
        # Draw the rectangle onto the Cairo context
        cr.rectangle(12, 434-brakedrawing, 94, brakedrawing)
        # Fill the rectangle
        cr.fill()

        # Throttle drawing
        cr.set_source_rgb(0.0, 1.0, 1.0)
        throttle = self.throttle
        throttledrawing=throttle*83.37 # multiply to scale the dimensions
        cr.rectangle(130, 434-throttledrawing, 94, throttledrawing)
        cr.fill()

        # RPM writing
        # set the font for the text
        cr.set_font_size(48)
        # Set the color for the text
```

```
cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color
# Draw the text
motor_speed = int(self.motor_speed) # max RPM value is 6500
(x, y, width, height, dx, dy) = cr.text_extents(str(motor_speed))
cr.move_to(265,257)
cr.show_text(str(motor_speed))

# Motor torque writing
cr.set_font_size(48)
cr.set_source_rgb(1.0, 1.0, 1.0)
motor_torque = int(self.motor_torque)
(x, y, width, height, dx, dy) = cr.text_extents(str(motor_torque))
cr.move_to(395,257)
cr.show_text(str(motor_torque))

# Vehicle speed writing
cr.set_font_size(100)
cr.set_source_rgb(1.0, 1.0, 1.0)
vehicle_speed = int(self.vehicle_speed)
(x, y, width, height, dx, dy) = cr.text_extents(str(vehicle_speed))
cr.move_to(335,190)
cr.show_text(str(vehicle_speed))

# DC current drawing
cr.set_source_rgb(1.0, 0.0, 1.0)
dc_current= self.dc_current
dccurrentdrawing=dc_current*0.384 # multiply to scale the dimensions
cr.rectangle(263, 414, dccurrentdrawing, 20)
cr.fill()

# DC current writing
cr.set_font_size(24)
cr.set_source_rgb(1.0, 1.0, 1.0)
(x, y, width, height, dx, dy) = cr.text_extents(str(dc_current))
cr.move_to(460,432)
cr.show_text(str(dc_current) + 'A')
```

```
        # DC voltage drawing
        cr.set_source_rgb(1.0, 0.0, 1.0)
        dc_voltage = self.dc_voltage
        dcvoltagedrawing=dc_voltage*0.27 # multiply to scale the dimensions
        cr.rectangle(263, 353, dcvoltagedrawing, 20)
        cr.fill()

        # DC voltage writing
        cr.set_font_size(24)
        cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color
        (x, y, width, height, dx, dy) = cr.text_extents(str(dc_voltage))
        cr.move_to(460,371)
        cr.show_text(str(dc_voltage) + 'V')

        #Motor voltage drawing
        cr.set_source_rgb(1.0, 0.0, 1.0)
        motor_voltage=self.motor_voltage
        motorvoltagedrawing=motor_voltage*0.27 # multiply to scale the
dimensions
        cr.rectangle(263, 291, motorvoltagedrawing, 20)
        cr.fill()

        # Motor voltage writing
        cr.set_font_size(24)
        cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white color
        (x, y, width, height, dx, dy) = cr.text_extents(str(motor_voltage))
        cr.move_to(460,310)
        cr.show_text(str(motor_voltage) + 'V')

        # Front left tyre temperature drawing
        fltyretemperature= 70
        if fltyretemperature<50:
            cr.set_source_rgb(0.0, 0.0, 1.0)
        elif 50<=fltyretemperature<=70:
            cr.set_source_rgb(0.0, 1.0, 0.0)
        elif 70<fltyretemperature<=80:
            cr.set_source_rgb(1.0, 0.647, 0.0)
```

```
elif fltyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(542, 57, 67, 99)
cr.fill()

# Front right tyre temperature drawing
frtyretemperature= 70
if frtyretemperature<50:
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=frtyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 70<frtyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif frtyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(719, 57, 67, 99)
cr.fill()

# Rear right tyre temperature drawing
rrtyretemperature= 70
if rrtyretemperature<50:
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=rrtyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 70<rrtyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif rrtyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(719, 291, 67, 100)
cr.fill()

# Rear left tyre temperature drawing
rltyretemperature= 70
if rltyretemperature<50:
    cr.set_source_rgb(0.0, 0.0, 1.0)
elif 50<=rltyretemperature<=70:
    cr.set_source_rgb(0.0, 1.0, 0.0)
```

```
elif 70<rltyretemperature<=80:
    cr.set_source_rgb(1.0, 0.647, 0.0)
elif rltyretemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(542, 291, 67, 100)
cr.fill()

# Battery pack temperature drawing
batterytemperature= 55
if batterytemperature<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=batterytemperature<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
elif batterytemperature>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(631, 219, 67, 58)
cr.fill()

# Controller temperature drawing
controller_temp = self.controller_temp
if controller_temp<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=controller_temp<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
elif controller_temp>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
cr.rectangle(631, 290, 67, 22)
cr.fill()

# Engine temperature drawing
motor_temp = self.motor_temp
if motor_temp<60:
    cr.set_source_rgb(0.0, 1.0, 0.0)
elif 60<=motor_temp<=80:
    cr.set_source_rgb(1.0, 1.0, 0.0)
elif motor_temp>80:
    cr.set_source_rgb(1.0, 0.0, 0.0)
```

```
        cr.rectangle(651, 326, 29, 54)
        cr.fill()

        # Warning messages writing
        cr.set_font_size(15)
        cr.set_source_rgb(1.0, 0.0, 0.0)  # set RGB values for red color
        if motor_voltage>600:
            cr.rectangle(263, 40, 248, 64)
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)  # set RGB values for white
color
            problem = "High engine voltage"
            action = "!STOP THE CAR!"
            (x, y, width, height, dx, dy) = cr.text_extents(problem)
            cr.move_to(325,65)
            cr.show_text(problem)
            (x, y, width, height, dx, dy) = cr.text_extents(action)
            cr.move_to(328,90)
            cr.show_text(action)

        elif dc_voltage>400:
            cr.rectangle(263, 40, 248, 64)
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)
            problem = "High battery voltage"
            action = "!STOP THE CAR!"
            (x, y, width, height, dx, dy) = cr.text_extents(problem)
            cr.move_to(325,65)
            cr.show_text(problem)
            (x, y, width, height, dx, dy) = cr.text_extents(action)
            cr.move_to(328,90)
            cr.show_text(action)

        elif dc_current>400:
            cr.rectangle(263, 40, 248, 64)
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)
```

```
        problem = "High battery current"
        action = "!STOP THE CAR!"
        (x, y, width, height, dx, dy) = cr.text_extents(problem)
        cr.move_to(325,65)
        cr.show_text(problem)
        (x, y, width, height, dx, dy) = cr.text_extents(action)
        cr.move_to(328,90)
        cr.show_text(action)

    elif brake<0.02:
        cr.rectangle(263, 40, 248, 64)
        cr.fill()
        cr.set_source_rgb(1.0, 1.0, 1.0)
        problem = "No brakes detected"
        action = "!STOP CAREFULLY!"
        (x, y, width, height, dx, dy) = cr.text_extents(problem)
        cr.move_to(325,65)
        cr.show_text(problem)
        (x, y, width, height, dx, dy) = cr.text_extents(action)
        cr.move_to(328,90)
        cr.show_text(action)

    elif batterytemperature>80:
        cr.rectangle(263, 40, 248, 64)
        cr.fill()
        cr.set_source_rgb(1.0, 1.0, 1.0)
        problem = "Battery overheating"
        action = "!COOL CAR DOWN!"
        (x, y, width, height, dx, dy) = cr.text_extents(problem)
        cr.move_to(325,65)
        cr.show_text(problem)
        (x, y, width, height, dx, dy) = cr.text_extents(action)
        cr.move_to(328,90)
        cr.show_text(action)

    elif controller_temp>80:
        cr.rectangle(263, 40, 248, 64)
```

```
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)
            problem = "Controller overheating"
            action = "!COOL DOWN CAR!"
            (x, y, width, height, dx, dy) = cr.text_extents(problem)
            cr.move_to(325,65)
            cr.show_text(problem)
            (x, y, width, height, dx, dy) = cr.text_extents(action)
            cr.move_to(328,90)
            cr.show_text(action)

        elif motor_temp>80:
            cr.rectangle(263, 40, 248, 64)
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)
            problem = "Engine overheating"
            action = "!COOL CAR DOWN!"
            (x, y, width, height, dx, dy) = cr.text_extents(problem)
            cr.move_to(325,65)
            cr.show_text(problem)
            (x, y, width, height, dx, dy) = cr.text_extents(action)
            cr.move_to(328,90)
            cr.show_text(action)


        elif fltyretemperature>70 or frtyretemperature>70 or
rltyretemperature>70 or rrtyretemperature>70:
            cr.rectangle(263, 40, 248, 64)
            cr.fill()
            cr.set_source_rgb(1.0, 1.0, 1.0)
            problem = "Tyre(s) overheating"
            action = "!TOO AGRESSIVE!"
            (x, y, width, height, dx, dy) = cr.text_extents(problem)
            cr.move_to(325,65)
            cr.show_text(problem)
            (x, y, width, height, dx, dy) = cr.text_extents(action)
            cr.move_to(328,90)
            cr.show_text(action)
```

```
# Queue for communicating the thread and dashboard GUI.
q = queue.Queue(2)

# Create and configure dashboard.
db = UTRONwindow(q)


# Create and start thread with the controller data reading and log the
information.
logger = UtronLogger(q)
logger.start()


# Start the application (main loop).
db.run()

# Stop the communication thread.
# This ensures the serial port is properly closed.
logger.stop()
logger.join()


# Translate the CSV file to an excel file.
excel= UtronExcel()
excel.translation()

print("Exiting")
```

UGRANOLLERS
UNIVERSITAT DE VIC
UNIVERSITAT CENTRAL
DE CATALUNYA