POLITECNICO DI TORINO

Master of Science in Automotive Engineering
A.Y. 2022/2023

Electrification and Control of a Scale Vehicle

Supervisor:
Prof. Alessandro Vigliani
PhD Angelo Domenico Vella
PhD Antonio Tota

Candidati:
Luca Biondo

July 2023

Contents

List of Figures

List of Tables

Abstract

The target of this research work is the electrification of a 1:5 scale vehicle used in off-road races allowing complete control on its dynamic, bypassing the stock transmitter/receiver system, poor in compatibility. The original vehicle is equipped with a 2-stroke internal combustion engine (ICE). It features a fully adjustable suspension system closely resembling those found on full-scale cars, including springs, dampers, and anti-roll bars, for both axles. Adjustable settings include the variation of spring's stiffness, damping response, suspensions hardpoints etc. Thanks to the vehicle reduced dimensions the setup could be quickly varied, unlike full-size cars, which requires significant effort and specific tools. Moreover, the size of the vehicle is significantly advantageous with regards to testing, since such a small vehicle does not require specialized test tracks and significant safety precautions. The electrification consists in the addition of a brushless DC motor, a planetary gearbox optimizing the speed/power range, an electric motor controller and a battery covering the role of power source provisioning, the selection criteria of these new components are provided. The layout of the assembly is designed to integrate the additional components. The manufacturing processes are well listed, e. g., the toothed gear slot creation to secure it to the planetary by means of a key, allowing the power transmission from the planetary to the ground, as well as the occurring chassis modifications made to accommodate the electric motor and the L-shaped mounts selection process. It is important to emphasize the reliance on Solidworks CAD models during this phase as they provide a comprehensive system overview, capable of being used for sizing, and verifying dimensions and volumes, which were otherwise challenging to measure. It is found that the new electric powertrain configuration is effective in space efficient. Several systems are used to monitor and control the powertrain speed, torque, and steering angle of the vehicle. The thesis lists and motivates the tests conducted for fine-tuning and controller optimization, moving among different control signals, both analogical than digital. The controller setting is done thanks to the ESCON Studio software. The signals are routed to the electric motor controller, what plays a crucial role in achieving a precise vehicle control, also giving an overview of the system status, resulting in consistent testing, reducing discontinuity sources weight, letting focus on the dynamic behavior. The early stage is a powertrain bench testing procedure in which an Arduino Portenta H7 board controls the system's speed, in charge of Pulse Width Modulation (PWM) signal production. The sketches development, by means of the Arduino IDE software and wiring connections are discussed. The Arduino board is a powerful and versatile instrument, easy to use for different tasks or projects, even if suffering from a lack of data management as a drawback. The following step moves to an analogical signal coming from a potentiometer, having Arduino implied in the speed requested and obtained comparison. In this way the analog to digital signal conversion, performed by Arduino, is avoided. In the advanced, the powertrain is offline tested, being mounted on the vehicle. The speed control is achieved using the PWM signal coming from the stock wireless transmitter/receiver system. Lastly the control is performed by means of the real-time target machine Speedgoat. This system, operating with real-time Simulink models, is employed for advanced control development, as it is capable of processing any kind of signal. It is used both for receiver output signals decoding, allowing to perform a detailed study, that for motor and steering servo stand-alone control. The cons behind this system are related to its dimensions and weight. In conclusion, this research successfully electrifies a 1:5 scale off-road racing vehicle, providing insights into the selection of components, modifications made, and control systems employed. The flexibility provided by these modifications and the precision achieved in vehicle control allows the electric vehicle to be very beneficial in the dynamic field having also potential for further advancements in this domain.

Introduction

The focus of this research work is the Electrification and Control of a four-wheel-drive 1:5 scale off-road racing vehicle.

The base vehicle is equipped with a 2-stroke 32 cc internal combustion engine (ICE), capable of producing a peak power of 2.4 kW and a peak torque of approximately 2 Nm. The speed of the ICE is regulated by an electric servo motor directly controlling the throttle valve. The steering relates on an Ackermann adjustable system, and, as before is controlled by means of an electric servo motor. The regulating signal for both speed and steering come from a stock transmitter/receiver system.

Figure 1. Losi 5ive-T 2.0.

The vehicle's suspensive system closely resembles that of a full-sized vehicle, allowing it to be ideal for lateral and longitudinal dynamic tests. This is composed by springs, dampers, and anti-roll bars for both axles. These components are fully adjustable, allowing to modify settings as spring stiffness, damping response, and suspension hardpoints. The advantages of utilizing a scale vehicle for testing purposes are numerous, including the ability to make quick setup modifications together with the requirement of a smaller testing area compared to full-scale vehicles.

The motivation for this project stems from the need to achieve consistent and precise results in controlling the off-road racing vehicle. The conventional transmitter/receiver system used in the base vehicle does not provide the necessary precision in setting relevant parameters such as speed and steering angle. Additionally, the vibrations generated by the internal combustion engine (ICE) hinder the utilization of sensors mounted directly on the chassis.

To overcome these issues, a potential solution is to move on an electric powertrain (ePowertrain). An ePowertrain offers greater controllability and is less sensitive to external factors. Accounting for this system, it becomes easier to set and adjust parameters using more sophisticated control systems. Similarly, the steering control can benefit from the improved precision and flexibility offered by an electric system. Having as a final result the achievement of consistent and precise test results, essential for future usage of the tuned vehicle.

Several control systems are employed throughout the research. Initially, Arduino, in the Portenta H7 version, a versatile board, is used for powertrain bench testing and early on-board tests, controlling the speed by means of Pulse Width Modulation (PWM) signals. The development of sketches using the Arduino IDE software and wiring connections is discussed. Additionally, an analog signal from a potentiometer is utilized for the purpose, avoiding the need for analog-to-digital signal conversion. Furthermore, the wireless control, by means of the stock transmitter/receiver system is performed and lastly, the real-time target machine Speedgoat, operating with Real-Time Simulink models, is utilized for advanced control development. It facilitates detailed study, standalone control of the motor and steering servo, along with processing of various signals.

The thesis is structured as follows:

- Preliminary Analysis: This section explains the process of selecting the components used in the electrification process. It considers devices such as the brushless DC motor, the planetary gearbox, the electric motor controller (Maxon ESCON 50/5), and the on-board power source (battery);
- Hardware: This chapter provides detailed explanations of the Arduino Portenta H7 board, the Maxon ESCON 50/5 motor controller, and Speedgoat, the real-time target machine. Detailing the components, the main features present, the interfaces, aiming to act as a quick start user guide, a review of the connection ports presents, pin in the case of Arduino, connector plugs for the controller, and a mix for the Speedgoat system. Explaining the wirings needed by the tests, the description of the sketches developed for the Portenta board, and the Simulink models needed by Speedgoat;
- Components Installation: here is described the process followed for the transition to the electric powertrain, including the modifications made to the chassis and the integration of the new components. It deals with the development of CAD models for accurate measurements and dimensional analysis;
- Experimental Tests: This one outlines the tests conducted to optimize the powertrain, explaining the controller setup, the devices used and the relevance of the test for the system tuning. It covers both bench tests and in-motion tests using different control systems. The discrepancies obtained during the potentiometer speed control, among the requested and obtained speed signals, are studied by means of MATLAB, what features an enhanced data plotting and analysis;
- Conclusions: This final section summarizes the findings of the research and provides insights into the successful electrification of the 1:5 scale off-road racing vehicle. It highlights the advantages of the modifications made, the precision achieved in vehicle control, and the potential for further advancements in this field.

In the conclusion section, the work is summarized and the main findings and outcomes are presented, together with some future development options.

Dynamics overview

Vehicle dynamics is a fundamental aspect in understanding and predicting the behaviour of a vehicle under different conditions. It spans among various components that contribute to the overall performance and handling characteristics of the vehicle. The key elements of the system include the vehicle body (or sprung mass), suspension components (such as springs and dampers), and tires (representing the unsprung mass).

In the study of vehicle dynamics, different modelling approaches have been developed to analyse and simulate the behaviour of vehicles. These models have evolved from simple lumped parameter models to more sophisticated Finite Element Method (FEM) and multi-body models.

Lumped parameter models provide a simplified representation of the vehicle's dynamics by considering the vehicle as a single mass, neglecting the individual characteristics of different components. These models are often used for preliminary analysis and quick estimations of vehicle behaviour.

On the other hand, more advanced modelling techniques, such as FEM and multi-body models, offer a more detailed and accurate representation of the vehicle dynamics. Finite Element Method (FEM) models consider the structural deformations of the vehicle body and analyse the effects of forces and loads on different components. This approach allows for a more comprehensive understanding of the vehicle's structural integrity and its impact on the overall dynamics.

Lastly, multi-body models, consider the interaction between multiple components of the vehicle, including the suspensions, steering system, and tires. These models consider the non-linearity of vehicle's components and enable a more realistic representation of the vehicle's behaviour under different operating conditions. They are particularly useful for studying complex manoeuvres, such as cornering, braking, and acceleration.

The evolution of vehicle dynamics models has also led to the development of non-linear models, which consider the non-linear relationships between inputs and outputs. These models are capable of capturing more realistic behavior, accounting for factors such as tire slip, suspension stiffness variations, and nonlinearities in the control systems. Non-linear models provide a more accurate representation of the vehicle's dynamic response and are widely used in advanced vehicle control and optimization.

Vehicle dynamics places significant emphasis on three fundamental rotational movements: yaw, pitch, and roll. These movements describe the rotation of a vehicle around its vertical, lateral, and longitudinal axes, respectively. Yaw, pitch, and roll play critical roles in shaping the vehicle's stability, handling, and manoeuvrability characteristics, and are essential considerations in the field of vehicle dynamics.

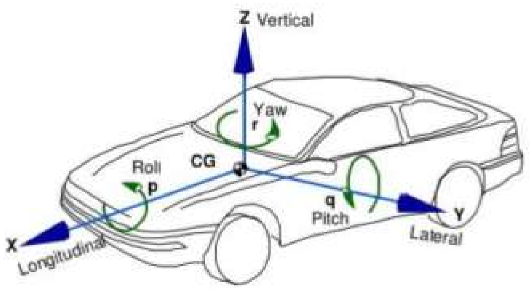


Figure 2. Vehicle Reference System.

In vehicle dynamics, yaw, pitch, and roll are terms used to describe the rotational movements around the vertical, longitudinal, and lateral axes of a vehicle. Yaw affects the vehicle's directional changes and stability during cornering, while pitch is influenced by acceleration and braking forces, impacting traction and weight transfer. Roll is the result of lateral forces during cornering and is crucial for maintaining stability and grip. Understanding, analyzing, and effectively managing these motions is essential in vehicle dynamics research and control, particularly through the implementation of an appropriate suspension system setup.

1. Preliminary Analysis

The first step into vehicle's electrification involves the selection of the required components to ensure a behaviour similar to the original configuration. To perform the substitution, the following ones are needed: an electric motor, a planetary gearbox, a controller, and a battery.

1.1. eMotor

The Losi 5ive 2.0 accounts for a two-stroke gasoline engine (Zenoah G320RC) capable of 2.4 kW maximum output power and a peak torque of 2Nm. According to that the focus is, in first approach, on something having a similar power. Along with considering mechanical constraints, particular attention is putted on the occupied room, the voltage required, which directly impacts on battery dimensions, and on the components' cost. The desired eMotor is a brushless DC one, for sake of controllability and cost-effectiveness.

A comparative table of various models of interest is being utilized for reference:

Figure 3. Zenoah G320RC.

Potential eMotor							
\	Tension/speed constant [kV]	Max Torque [Nm]	Power [W]	Type	Voltage [V]	Price [€]	Link
ICE	\	2	2400	\	\	\	\
Skateboard Motor	170	\	3000	DC Brushless	24	107.09	link
Alomejor	120	\	4600	DC Brushless	24/12	109.68	link
DiLiBee Outrunner	120	2.4-4.0	2970	DC Brushless	24/60	79.99	link
L-faster	\	\	350	DC Brushless	24/36	91.99	link
EG-BIANSU	\	1.15	130	DC Brushless	24	133.39	link
EG-BIANSU	\	0.56	105	DC Brushless	24	103.35	link
DOGA	\	0.2	63	DC Brushless	24	95.99	link
Brushless RS PRO	\	0.75	500	DC Brushless	24	130.41	link
EMP N5065	270	0	1820	DC Brushless	24	85	link

Table 1. Potential eMotor.

The RS PRO is the eMotor what better fit all the requests, including a Hall Sensor for a closed loop control strategy too. It is a 24V brushless DC motor, with a maximum current of 15A peaking at 500W power and 0.75Nm torque. Aiming to modulate and improve the produced torque a planetary gearbox is then added.

The following picture explains the wiring colour scheme, useful for the connection of it.

Figure 4. RS PRO Brushless Motor.

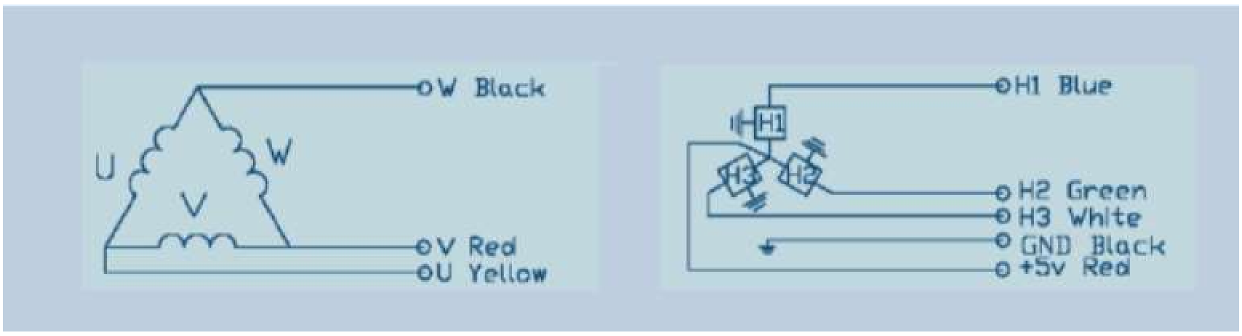


Figure 5. eMotor wiring scheme.

1.2. Controller

The controller is a device required for eMotor management. The device performs the tasks of phase management, energizing them in the proper sequence, energy management, current regulation, and feedback control using the Hall sensor integrated into the motor.

The chosen controller is the Maxon Motor Escon 50/5, which offers a range of features. It allows for managing PWM or analog signals, controlling current and speed in both closed and open loop configurations, providing a wide selection of output values from the system. Additionally, it can handle motor feeding voltage levels ranging from 10 to 50V, ensuring adaptability and potential use in future projects.

An in-depth review of this devis is conducted in the Maxon Escon Controller chapter.

1.3. Planetary Gearbox

After choosing the eMotor, it is needed to select a planetary gearbox to increase the peak torque of the eMotor, which is 0.75Nm.

As for the eMotor a selection of devices for the speed/torque management has been made, taking into consideration the room occupied by the device, the cost, and the installation screws position, trying to exploit the holes already existing on the eMotor face.

Potential Gearboxes						
\	Gear Ratio	Input Shaft Diameter [mm]	Output Shaft Diameter [mm]	Nema Standard	Price [€]	Link
Planetary Gearbox	**5:1**	**8**	**14**	**Nema 23**	**133**	link
Cambio planetario	**5:1**	**8**	**14**	**Nema 23**	**73**	link
Riduttore di velocità	**5:1**	**8**	**14**	**Nema 23**	**94**	link
Planetary Gearbox	**4:1**	**8**	**14**	**Nema 23**	**78.40**	link
Riduttore Planetario	**4:1**	**8**	**14**	**Nema 23**	**119**	link
Nema 17 riduttore	**5:1**	**5**	**8**	**Nema 17**	**64**	link
Nema 23 riduttore	**5:1**	**6.35**	**14**	**Nema 23**	**70**	link
Nema 17 riduttore	**5:1**	**5**	**8**	**Nema 17**	**46**	link

Table 2. Potential Gearboxes.

The searched gear ratio was 4:1 or 5:1, in this way the peak torque raised up to 3Nm or 3.75Nm. The gearboxes follow the Nema standard which provides the positions of the mounting screws and the input/output shafts diameter. According to the 5mm eMotor output shaft the last row, Nema 17, reducer is selected.

Since the eMotor mounting screws does not follow a Nema standard in the ePowertrain Assembly chapter the mounting process is explained.

1.4. Battery

The final component required for the ePowertrain is an energy storage system, specifically a 24V battery that matches the voltage requirement of the eMotor. Since the gasoline tank is no longer necessary, the space previously occupied by the tank is now available for accommodating this device.

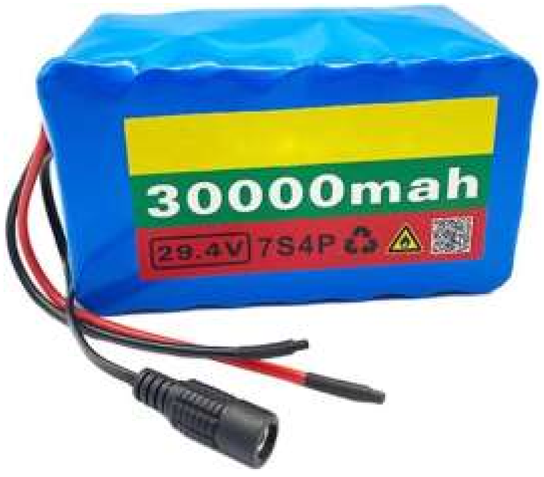


Figure 6. Selected Battery

The selection was based on factors such as the peak current of 20A and the price, considering the best trade-off in terms of space, weight, and capacity.

As for the previous cases the following table aims to summarize the main solution found. Some manufacturers are able to design a proper battery pack, according to the needs.

Potential Batteries						
\	**Tension [V]**	**Capacity [Ah]**	**Weight [Kg]**	**Dimensions [mm]**	**Price [€]**	**Link**
Batteria Litio	**24**	**10**	**2**	**95x55x125**	**148**	link
Batteria Litio	**24**	**30**	**1.6**	**130x80x66**	**122.62**	link
Battery pack Seilylanka	**25.9**	**6**	**0.7**	**68x38x130**	**84.99**	link
Battery pack Seilylanka	**24**	**12**	**1.4**	**38x68x260**	**135.99**	link
Batteria al litio F1C4	**24**	**10**	**1.75**	**140x80x75**	**149.99**	link
Battery pack DBA SRL	**24**	**12.8**	**3 (Stimato)**	**80x150x85**	**229.50+Iva**	
Battery pack mr-electric	**24**	**13**	\	\	**135+20% fattura e 25€ caricabatterie**	
Battery pack ilRicaricabile	**24**	**5.8**	**0.7**	**130x50x70**	**150€+iva**	
Battery pack Batteryclinic.it	**24**	**5.8**	**0.7**	**130x50x70**	**95+iva e 50€ caricabatterie**	
Battery pack Batteryclinic.it	**24**	**7**	**0.7**	**130x50x70**	**130+iva e 50€ caricabatterie**	

Table 3. Potential Batteries.

The provided, by-hand, technical drawing illustrates the tank's dimensions, which are crucial for estimating the required space and accommodating the selected Li-Ion battery.

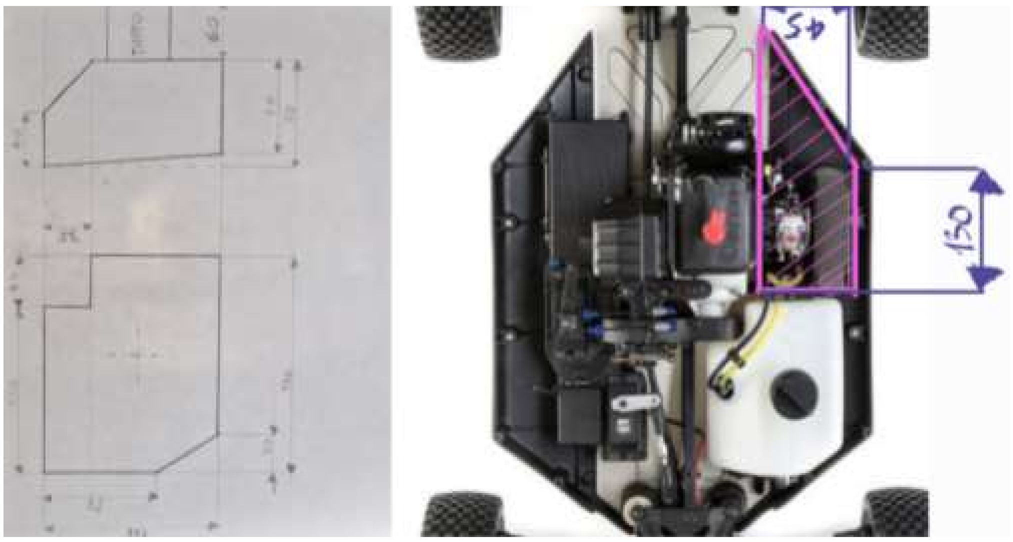


Figure 7. Tank Technical Drawing.

2. Hardware

2.1. Arduino

2.1.1 Introduction

The *Arduino Portenta H7* is a board belonging to the Arduino family. These devices are programmable boards able to accomplish a variety of tasks, interacting with external environment through some connecting pins used to communicate with such as: LEDs, potentiometers, motors, displays, but not only, it is possible to obtain analogical or digital outputs directing information for each kind of control logic.

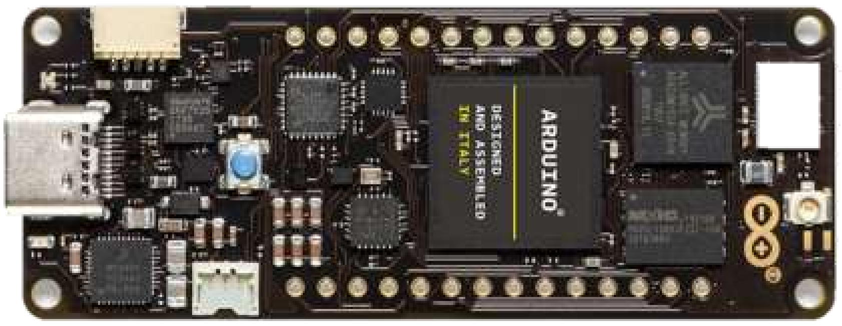


Figure 8. Arduino Portenta H7 Board.

These boards could be used stand-alone or to design a final, more complex, projects. The pins are very helpful in the early steps of project design: they could be connected without the needs of being soldered, allowing the connections modulation and addition in devices with no much effort; furthermore is possible to avoid the printing of a new PCB each time. Each board is equipped with a microprocessor, a RAM, a ROM, a DAC, an ADC, the feature of producing a PWM signal and so on.

The *Portenta* boards family are designed to accomplish heavier tasks with respect to the standard Arduino boards, in fact them includes a more powerful, dual core, processor what allows to execute operations in parallel. It is named STM32H747, working with high level codes. In addition to that, a GPU, the Chrom-ART Accelerator™, is present, useful in case of external monitor interface. Lastly, for what concerns the SOC, is the presence of a dedicated JPEG encoder and decoder.

The logic circuit works at 3.3 Volts, while the board could be fed up to 5 Volts.

2.1.2 Characteristics

The main features will be discussed in the followings, a complete collective datasheet is available here.

Processor

The processor built on this board is the STM32H747 - dual core processor, composed by an Arm® Cortex® M7 running at 480 MHz and an Arm® Cortex® M4 running at 240 MHz. Both cores have a 32-bit architecture and have all access to the peripherals without any restriction. Thanks to that, it is possible to execute parallelised operations, one for each core.

It is better to denote that the first one is the most precise and powerful, not just for the higher clock achievable, but considering the architecture and the L1 cache, not included in the second one.

Connectivity

Pins

The Portenta H7 is equipped with 28 pins. Differently from other Arduino boards, the Portenta H7 does not include a pin header, it could be added if needed. A variety of options are presents in the market, according to the needs of the application; in Figure 9 is shown the pin header soldered to the used Portenta board.

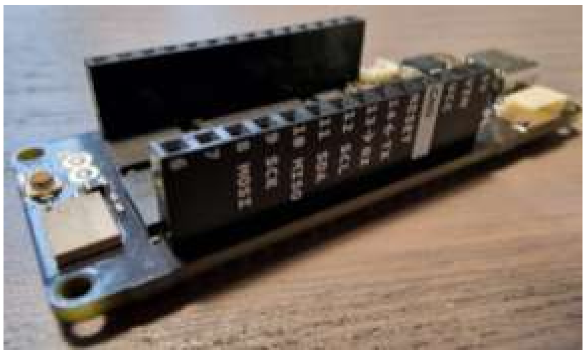


Figure 9. Arduino Portenta H7 Soldered Headers.

The built pins are "holes" in which is possible to solder the wires, alternatively you can fit and twist them, what results to be very helpful during the first steps of a project where modifications in the connection scheme are more frequent.

Each pin has a specific name, well denoted into the datasheet of the board. These names are used during the code writing phase to specify which pin will perform the action. The pins could be digital, setting only binary values, or analogical, so exploiting a tension interval.

In the Figure 10 is possible to have an example of connected devices, in this specific case a potentiometer and two external LEDs are plugged; the red cable is linked to the GND pin, in this way is possible to merge negative sides of the devices into a single common wire; furthermore the RGB built LED is turned on in red, the power comes from a PC, by means of the USB cable.

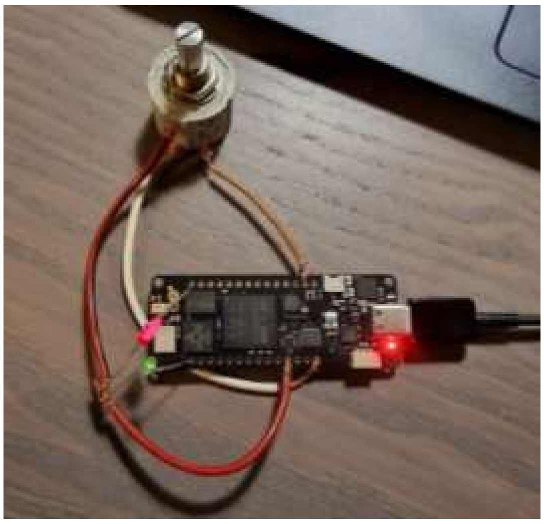


Figure 10. Example of external devices connected.

High density connectors

This board, in addition to the standard pins, accounts for two 80 pins *High Density Connectors* through which is possible to have connections in a compact way.

Arduino developers designed some components, called *Breakout Boards,* aiming to exploit the ports available through them. However, a dedicated component could be designed autonomously.

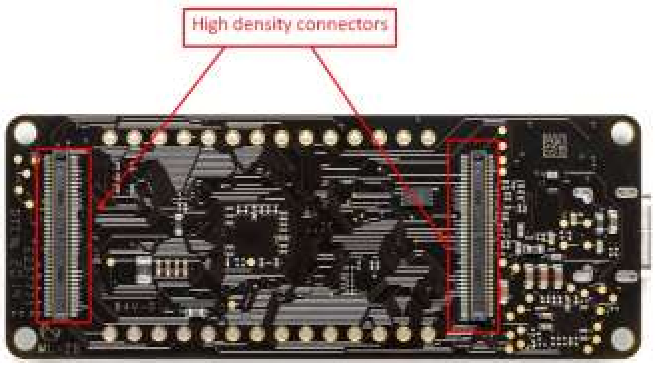


Figure 11. High Density Connectors.

USB Type-C

This module includes an USB Type-C connector, used to upload the written code and, optionally, power the board.

I^{2}C connector

An I^{2}C connector is present, it is used to link devices in a Master-Slave relation, allowing them to communicate; a possible application could be the one aiming to connect different Arduino boards for data sharing.

Wireless Connectivity

A very powerful feature of wireless communication is added. This is possible thanks to an included WiFi/Bluetooth® module. The board could connect to an existing wireless network or could be used as an access point. The Bluetooth® module could work in classical or BLE mode.

It is worth noting that the sketch still needs to be uploaded via the USB.

2.1.3 Power the Arduino

The voltage levels are internally regulated thanks to some LDOs (Low Dropout linear regulators).

The Arduino board automatically executes the last uploaded sketch in loop when an electric source is plugged. To stop and reset the Arduino board the power supply should be detached, otherwise, the small blue button does it without any disconnection, executing the whole code from the beginning, so both the single set instructions than the loop one.

Input

The Portenta H7 could be powered in different ways, the voltage must range between 3.7V and 6V.

USB

It is possible to energize the system via the USB port (5V, 3A), linking a cable from a computer, a USB wall charger, or a power bank.

Pins

The board presents a VIN and a GND pins, to which connect the positive and the negative sides of the power supply. VIN and GND stands for "Voltage Input" and "Ground". Alternatively, it is possible to supply power through the High-Density Connector.

Battery

Lastly, it is possible to connect a 3.7V Li-Po battery (having a minimum capacity of 700 mAh) to the J4 connector, the battery will be recharged if a connection via USB or pins occurs.

Output

The Arduino board is able to provide power to connected devices through the 5V pin, with not more than 3A or through the 3V3 pin, which provides a 3V level and a maximum of 500mA of current.

2.1.4 Coding Environment

Some tools must be implied to manage and upload codes needed by the board. There are several of them, but the greatest part requires a high level of consciousness in programming or are a pay-per-use software; some alternatives are: *Atom.io + Platformio, Eclipse for Arduino, Visual Studio with Arduino* etc.

Even if the upper cited software includes a bunch of smart features what helps in developing codes, they're designed for an advanced user, as a beginner, it is better to start with a more intuitive free-to-play software: Arduino IDE.

Arduino IDE

Arduino IDE, where IDE stands for "Integrated Development Environment", is an open platform developed by the Arduino team. Even if in some cases it does not result the smarter environment it is in constant improvement, thanks to the community support.

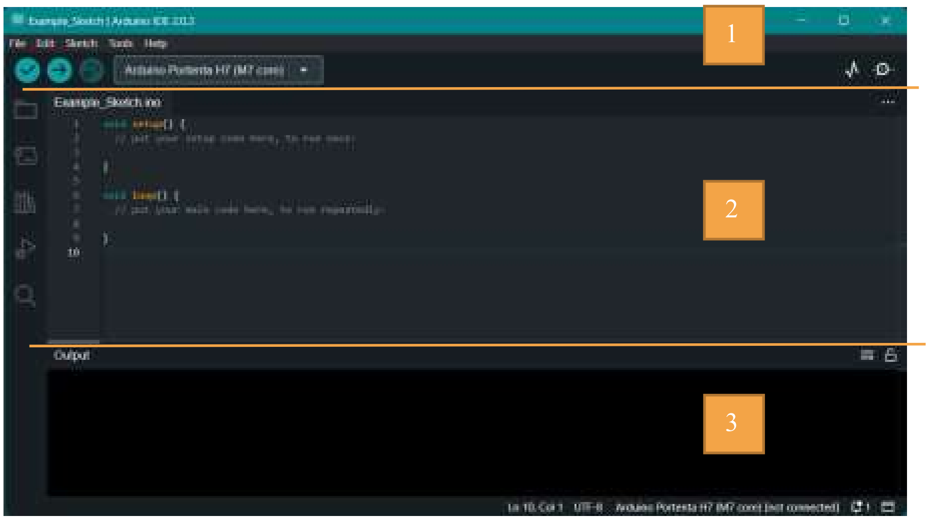
Arduino must be coded with C/C++ languages, the IDE features syntax highlighting, brackets control and automatic indentation, pay attention on capital letters usage being a case sensitive language.

The sketches will be saved in the *.ino* format and must be located into a folder named as themselves to work.

Interface

The IDE interface is quite clean and is possible to divide it into three main parts.

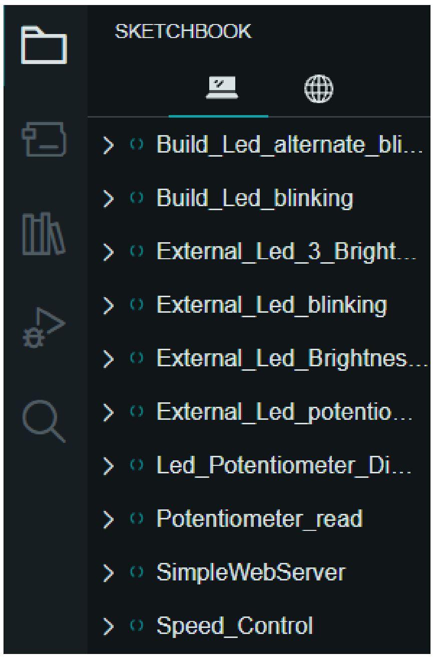
In the top area one, denoted with number 1, some menus are present, each of them includes a lot of useful functions for the script management, like: the debug option, the update of the software tab, the help feature, the settings menu, which also allows to modify graphic aspects, etc. It is worth nothing that is possible to have a look at many pre-defined scripts in the *Examples* section, helping in the first approach to the Arduino word. The buttons on the second row, are, in order from the left, the *verify* one, to check the presence of syntax errors, the *upload* one, to send the code to the selected board (in this case the Portenta H7 M7 Core is the chosen one), the *debug* one, a board selector drop-down menu, the *serial plotter* and *serial monitor* ones.


Example_Sketch | Arduino IDE 2.0.3
File Edit Sketch Tools Help
Arduino Portenta H7 (M7 core)
Example_Sketch.ino
1
2
Output
3
Ln 10, Col 1 UTF-8 Arduino Portenta H7 (M7 core) [not connected]

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 12. Arduino Interface.

Moving to the second section, on the left, is possible to find some shortcuts, from the top: the *sketchbook,* in which is possible to have quick access to all the scripts included into the path, as shown in Figure 13; then, the *boards manager*, where is possible to download drivers for a certain hardware and to update them; the *library manager*, in which is possible to find the already installed libraries, looking for updates, or to download the needed ones (a library is a set of specific functions); following there is the *debug* one and lastly the *search* shortcut, needed for finding text (like variables or functions) in long scripts.

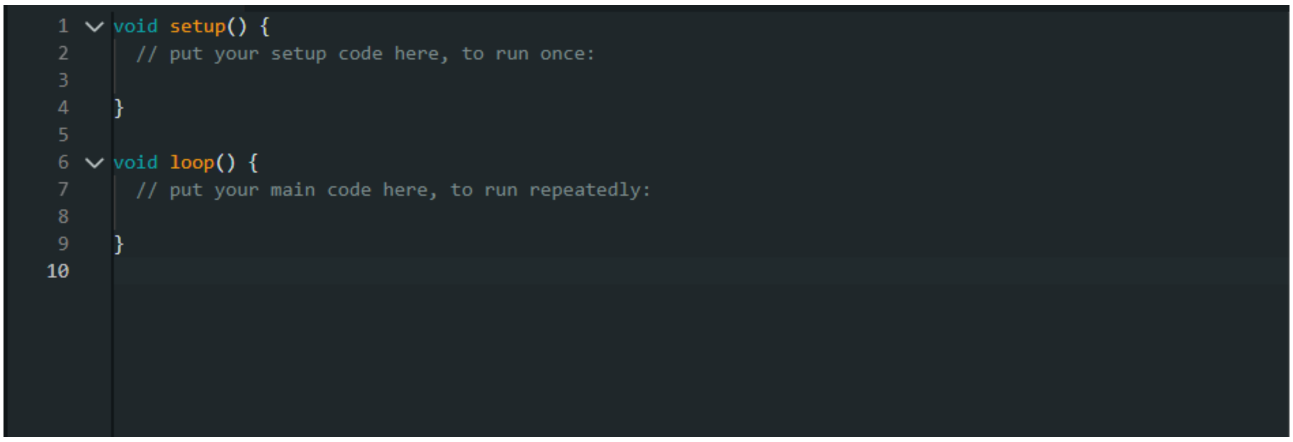


Build_Led_alternate_bli...
Build_Led_blinking
External_Led_3_Bright...
External_Led_blinking
External_Led_Brightnes...
External_Led_potentio...
Led_Potentiometer_Di...
Potentiometer_read
SimpleWebServer
Speed_Control

Figure 13. Sketchbook.

The third and last section include the outputs information. Once a sketch is uploaded to the board, in this box is possible to check the status of the operation, divided into *erase* and *upload* phases, some information about occupied memory, given errors and data coming from the board, thanks to the serial monitor interface. Differently, the *serial plotter* opens a new window in which a graphic representation of the selected values occurs.

Sketch box



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 14. Sketch box.

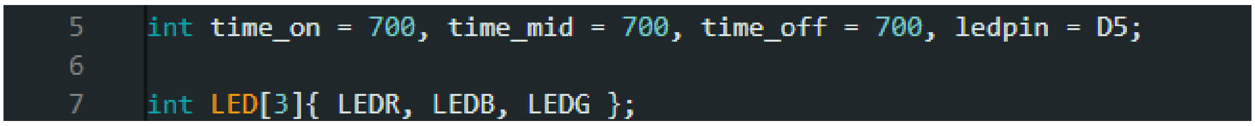
The *sketch box* is the place to write the code. In it there are two main functions, by which the code is composed: the upper one is executed only one time by the board, due to that, here only the instructions related to setup, non-varying functions must be added, for instance, is possible to turn on a LED, without any change, for the whole execution time in order to check the status of the system; in contrast, the second function is the one what will be executed in loop and stops only when the power is disconnected from the board (alternatively the reset button is pushed), for instance is possible to blink a LED by turning it ON and OFF periodically. Each code row must be followed by a semicolon terminator.

2.1.5 Instructions

The sketches are composed by two main elements: variables and functions.

Variables

The variables aim to store data, like numbers or characters. They need to be declared at the beginning of the script, assigning to each one a type, in function of what, the compiler locates a certain amount of space in the memory, the types are a few, the main ones are: *int, float, double, char*.



```
int time_on = 700, time_mid = 700, time_off = 700, ledpin = D5;

int LED[3]{ LEDR, LEDB, LEDG };
```

Figure 15. Variables definition.

The Figure 15 is an example of what said, the first row defines 4 integer variables, assigning a number to them too, apart from the last one where D5 is a pin name what will be translated by the compiler into an integer number thanks to the library. In the second row, instead, a vector of 3 elements is built, the vector name is LED, the content, as in the previous case is translated by the compiler into integers.

Functions

Unlike variables, functions are a set of instructions to complete certain tasks. There are different kinds of them, each library added to our code brings some new specific functions. Let's see some of the most common ones.[1]

pinMode(pin number, mode), this function is needed to specify how the pin will be used, if in input or in output; in case the pin is an analogical one it is already set as INPUT, thus the function could be avoided;

digitalWrite(pin number, mode), it is used to assign a digital value to the pin selected, the modes are HIGH and LOW, the first one implies the high voltage level of the pin, the latter, the low voltage level; the voltage levels are 3.1V for HIGH and 0V for LOW; following this, if it is wanted to turn on an external LED, it must be set on the HIGH mode, the opposite is valid;

digitalRead(pin number), this is the dual of the upper function, instead of setting an HIGH or LOW value, this function reads and gives back it; it works properly only with digital pins;

analogWrite(pin number, value), this one works in different ways according to the selected pin; some digital pins are able to produce a 50Hz PWM signal (the frequency could reach 240MHz adding the *Portenta_H7_PWM* Library), in this case the second input of the function, the *value*, is included in the range 0-255, corresponding to 0-3V, since it exploits 8 bits by default (what could be increased up to 32 bit); by contrast, thanks to the built DAC, through the A0 pin is possible to obtain an analog output value. As for the case of the PWM, the default resolution is 8 bits, but, if necessary, it could be enhanced to 12 bits, including *AnalogOut.h* library, the max voltage output is 3.1;

analogWriteResolution(value), in the case a more precise output is required, this function allows to change the *analogWrite()* resolution, according to the board characteristics. The *value* to be sent is just the desired bits resolution;

analogRead(pin number), this function is suitable only through analog pins (the ones named with an "A" letter before the number, like A0, A1…); by using this, the board is able to read a values ranging from 0V to 3V by default, the upper threshold is lowerable by connecting a voltage reference to the AREF pin; the function gives back a number included into the 0-1023 range, since the default resolution is set to 10 bits; the Arduino Portenta H7 includes a 16 bits ADC so the resolution could be set according to that;

analogReadResolution(value), as for the *analogWriteResolution()*, this one is needed to change ADC resolution, still taking into account the board limitations;

delay(number), by using this command, apparently, we pause the programme for the amount of specified time, in milliseconds, actually the CPU is still working with its own clock, but just doing empty cycles;

Serial.print(data, format), prints data to the serial port as ASCII text, while numbers are sent as numbers, if non integer are rounded, by default, to the second decimal; the second parameter is optional and includes a series of formatting options;

Serial.println(data), it works as the previous one, just including a new line after the printed text;

[1] A complete description of functions and syntax is available here.

Serial.begin(speed), it sets the data rate in bits per second (*baud*) for serial data transmission, mandatory to communicate with the serial monitor; notice that this value could be modified according to the needs, in that case the same value should be selected into the drop-down menu of the serial monitor;

map(number, actual range lower limit, actual range upper limit, new range lower limit, new range upper limit), this function is used to scale a *number* from a range to another one.

Further Syntax

Together with these functions some useful commands and structures are available, here are listed the most relevant. Be careful that, in general, commands must not be followed by a semicolon terminator.

`#define constantName value`, this is a very useful command, it is possible to assign a name to a constant value, the differences with the definition of a constant, are related to the fact that it does not require any space from the memory, it is a pure substitution of a value when a certain name is called, having, as a result, an increase in code speed and less occupied memory. The terminal semicolon must not be added;

`#include <LibraryFile.h>`, is used to add outside libraries in the sketch, and like the *define,* no semicolon must be added;

```
for (initialization; condition; increment) {
// statement(s);
```

`}`, this statement is used to repeat a set of instructions enclosed in brackets; the *initialization* happens just once and in general is a reset of a counter; if the *condition* is verified the loop is still performed, on the contrary, the execution moves to the following instruction; the *increment* is executed each time while the loop is repeated;

```
if (condition1) {
  // do Thing A
}
else if (condition2) {
  // do Thing B
}
else {
  // do Thing C
```

`}`, the *if* structure allows to perform some instructions if a certain condition is verified. In negative case it moves to the *else* function, if present, otherwise it proceeds with the following statements;

```
switch (var) {
  case label1:
    // statements
    break;
  case label2:
    // statements
    break;
  default:
    // statements
    break;
```

}, this command compares the value of the *var* with the different *labels*, in case of a match, the code executes the instructions under that specific one; once executed the instruction, it proceeds with the codes next to the *break*;

`// text` , by using these two slashes is possible to add a comment, all the text following, till the end of the row, will not be part of the instructions and could be used as a personal reference.

2.1.6 Sketches for Arduino Study

The following section presents a series of simple sketches designed to enhance familiarity with the programming environment and board features. These sketches utilize the built-in LED, external LEDs, and a potentiometer connected to the pins. It is worth noting that the built-in LED of the Arduino board used in this project is an RGB type, allowing for colour switching and mixing. However, it should be mentioned that this particular LED cannot be dimmed.

The board LED behaves in a different way with respect to the external ones. In fact, in order to turn ON the external LEDs, it is needed to set the pin voltage level to the HIGH one, instead, the OFF condition is obtained by setting the LOW voltage level. The internal one, on the contrary, turns ON with the LOW voltage level and OFF with the HIGH one; is possible to prove that uploading the Dual Behaviour Internal and External LEDs code.

In the Figure 16 the different colours are shown, independently and all together.

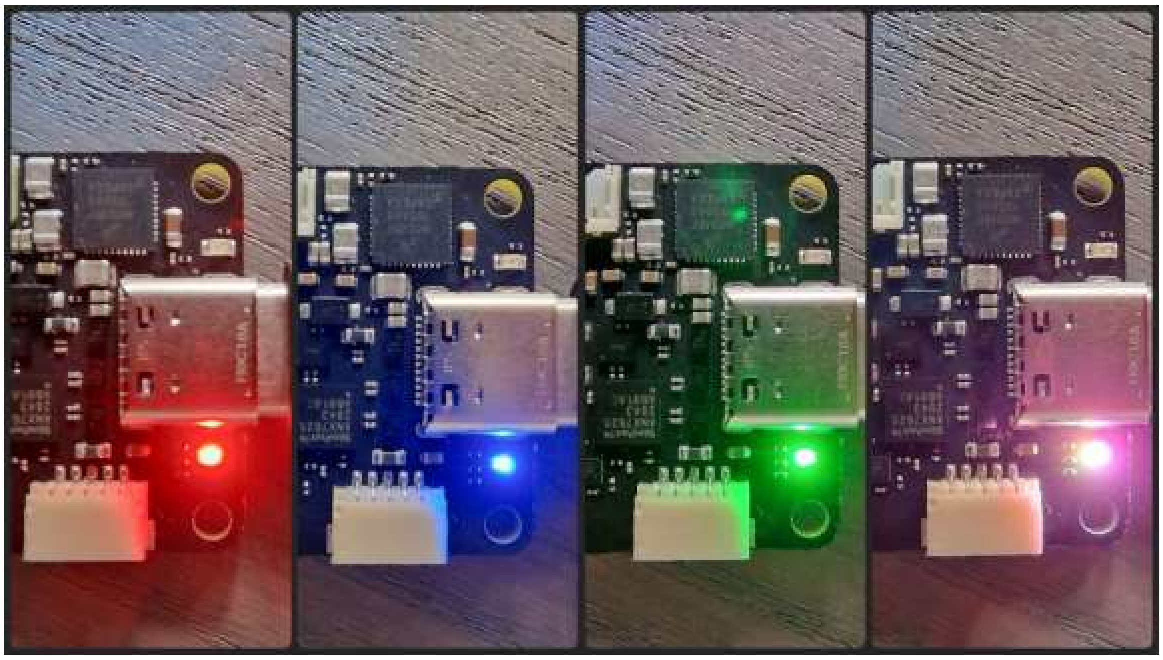


Figure 16. Built RGB LED.

Built LED Blinking

```
// Blinking built LED
int time_on = 800, time_off = 500;
int ledpin = LEDB; //you can choose among LEDB LEDG LEDR
int ON = LOW, OFF = HIGH;
```

```
void setup() {
  pinMode(ledpin, OUTPUT);
  digitalWrite(ledpin, OFF); //turn off the LED in case it was not
}

void loop() {
  digitalWrite(ledpin, ON); //turns on the LED
  delay(time_on);

 // digitalWrite(ledpin, OFF); //turns off the LED
  delay(time_off);
}
```

In this code the built LED is turned on in blue, for 800 ms and off for 500 ms. It is possible to have a more readable code associating the ON status to the LOW voltage level, as done by defining an integer variable, since this could be counter intuitive.

Built LED Alternate Blinking

```
// This sketch will mix built led colours
int time_on_g = 100, time_on_b = 100,time_on_r = 500, time_off = 100; //setting the ON time of
//the different colours
#define ON LOW
#define OFF HIGH

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);

}

void loop() {
  digitalWrite(LEDB, ON); //turns on the blue led
  delay(time_on_b);

  digitalWrite(LEDG,ON); //mix with the green one
  delay(time_on_g);

  digitalWrite(LEDR, ON); //mix with the red one
  delay(time_on_r);

  digitalWrite(LEDR, OFF); //turn off the red led
  delay(time_off);
}
```

In this code the different colours of the built LED are turned on with a certain delay, followed by turning off only the red one. Being the operation in the loop, the red colour results to blink. It is shown how to associate ON/OFF to LOW/HIGH without allocating memory by exploiting the *define* command, instead of variables definition.

Built LED Alternate Blinking 2

```
// This sketch turn on and off the 3 colours of the RGB LED sequentially
#define ON LOW
#define OFF HIGH
int LED[3]{ LEDR, LEDB, LEDG }; //all the LED colours are added to a single vector of
dimension 3
```

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {

  for (int i = 0; i < 4; i++) { //this for cycle turns ON sequentially the colours delayed by
//200ms
    digitalWrite(LED[i], ON);
    delay(200);
  }

  digitalWrite(ledpin, OFF);
  delay(time_off);
  for (int i = 0; i < 4; i++) { //this for cycle turn OFF sequentially the colours delayed by
//200ms
    digitalWrite(LED[i], OFF);
    delay(200);
  }
}
```

This sketch is similar to the previous one, but the LED colours are grouped into a single vector and a *for* cycle is exploited to have a more compact and readable code.

Dual Behaviour Internal and External LEDs

```
// Verify the dual behaviour of internal and external LEDs
#define ON HIGH
#define OFF LOW
#define ledpin D6 //the external LED is attached to the D6 port

void setup() {
  pinMode(ledpin, OUTPUT);
  }

void loop() {
  digitalWrite(ledpin, ON); //turns the external LED ON
  digitalWrite(LEDB, OFF);  //turns the internal LED ON

}
```

This code is built to show the opposed way to turn on the internal and the external LEDs, using this both LEDs turns ON, demonstrating that the HIGH voltage level only light the external LED and not the built one. The Figure 17 is the hardware configuration obtained thanks to the upper code.

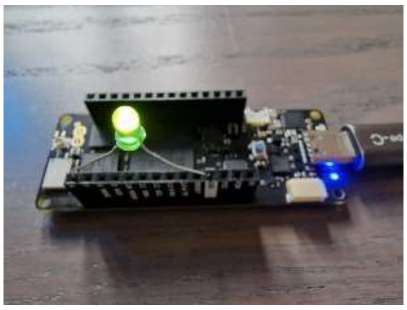


Figure 17. LEDs dual behaviour.

External LED Blinking

```
// External led control
#define ON HIGH
#define OFF LOW
int ledpin = D6, time_on=400, time_off=400;

void setup() {
  pinMode(ledpin, OUTPUT);
}

void loop() {
    digitalWrite(ledpin, ON);
    delay(time_on);

    digitalWrite(ledpin, OFF);
    delay(time_off);
}
```

This code aims to blink an external LED attached to the D6 pin.

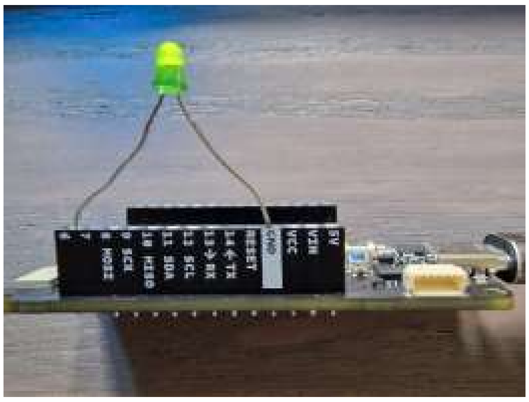


Figure 18. External LED D6 connected.

External LED Brightness Variation

```
// Regulate the led brightness by applying a PWM
int time_on = 1000, time_off = 1000, ledpin = D5; //remember to connect the long leg to the
positive connection

void setup() {
  pinMode(ledpin, OUTPUT);
}

void loop() {
  analogWrite(ledpin, 255); //in this case set the maximum value of the PWM is set (0-255
range, 8 bits resolution)
  delay(time_on);

  analogWrite(ledpin, 50); //50 is, more or less, the 20% of the duty cycle
  delay(time_off);
}
```

This code exploits the PWM feature to regulate the external LED brightness on two levels, the first is 100% duty cycle, while the second one is about the 20%. Since the standard 8 bits resolution is not modified, the range is 0-255. To exploit the PWM a PIN able to do so must be used.

External LED 3 Brightness Levels

```
// Setting the external LED on 3 brightness levels
#define ON LOW
#define OFF HIGH
int time_on = 700, time_mid = 700, time_off = 700, ledpin = D5;

void setup() {
  pinMode(ledpin, OUTPUT);
}

void loop() {

  analogWrite(ledpin, 255);  //the PWM range is 0-255
  delay(time_on);

  analogWrite(ledpin, 110);  //the PWM range is 0-255
  delay(time_mid);

  analogWrite(ledpin, 0);    //the PWM range is 0-255
  delay(time_mid);
}
```

This one is another usage of the PWM feature, used for moving through different brightness levels. The external LED brightness is decreased in 3 steps, accounting for the same time delay.

In this case the LED is linked to the D5 position, which allows to produce a PWM signal too.

Potentiometer Reading

```
//In this case the potentiometer values are read and printed to the serial monitor
#define ON LOW
#define OFF HIGH

int analogvalue;
int potpin = A0;  // The pin where the potentiometer is attached
int readvalue;

void setup() {
  Serial.begin(9600);      //data bits rate used to communicate with the monitor
  pinMode(potpin, INPUT);  //choose the analog pin to be used as input
}

void loop() {

  readvalue = analogRead(potpin);  // the read value is included into the 0-1023 range, 
default 10 bits resolution
  analogvalue = map(readvalue, 0, 1023, 0, 3.3); //scaling the value to the read voltage
  Serial.println(readvalue);

  switch (analogvalue) { //switch to selectively turn on one led colour
    case 0:
      digitalWrite(LEDR, ON);
      digitalWrite(LEDB, OFF);
      digitalWrite(LEDG, OFF);
      break;

    case 3.3:
      digitalWrite(LEDR, OFF);
      digitalWrite(LEDB, ON);
      digitalWrite(LEDG, OFF);
      break;

    default:
      digitalWrite(LEDR, OFF);
      digitalWrite(LEDB, OFF);
      digitalWrite(LEDG, ON);
  break;
  }
  delay(200);
}
```

In order to perform this sketch, a potentiometer needs to be connected to the board. This last device is able to support up to 5V, the board instead, reads analogical values only up to 3.3V. Accounting for that, higher values are saturated to the board maximum. The potentiometer has 3 connectors, the upper ones are the positive and the negative connectors, there is no specific voltage direction, the lower one is the regulated voltage signal. The power needed by the device is taken from the 5V pin of the Arduino board, the ground is connected to the board GND while the signal to the A0 pin.

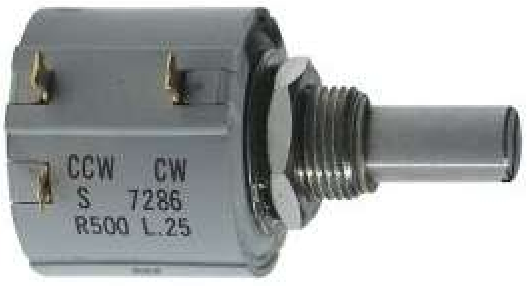


Figure 19. Potentiometer.

The read signal is scaled to the voltage level and printed to the serial monitor, after that, thanks to the *switch* command, the internal LED turns on in a specific colour according to the voltage level.

No resolution variations of the ADC are set, so the read value range is still 0-1023.

External LEDs switch

```
// External LEDs switched by the potentiometer
#define ON HIGH
#define OFF LOW
int ledpin = D5, ledpin2 = D6, potpin = A0;
int readvalue;

void setup() {
  pinMode(ledpin, OUTPUT);
  pinMode(ledpin2, OUTPUT);
  digitalWrite(LEDB, ON);
}

void loop() {
  readvalue=analogRead(potpin); //the obtained value is included in the 0-1023 range
  Serial.print("Value: ");
  Serial.print(readvalue);

  if (readvalue<=511) { //in case the voltage level is equal or minor the half interval the
led //attached to D5 is turned on and the one in D6 turned off
    digitalWrite(ledpin, ON);
    digitalWrite(ledpin2, OFF);
    Serial.print(", LED 1 ON - LED 2 OFF");
  } else {
    digitalWrite(ledpin, OFF);
    digitalWrite(ledpin2, ON);
    Serial.print(", LED 1 OFF - LED 2 ON");
  }

  Serial.print("\n \n");
  delay(200);
}
```

To run this sketch a potentiometer and two external LEDs must be implied, as in the Figure 10.

The aim of this sketch is to turn on one of the two external LEDs according to the signal coming from the potentiometer, if it is lower than half interval, 1.65V, the LED attached to the D5 pin is turned on, differently the one on the D6 pin is enabled. The built LED is turned on to verify the board safety.

Dimming external LED through the potentiometer

```
// Regulate the led brightness using the potentiometer read value
#define ON HIGH
#define OFF LOW
int ledpin = D5, potpin = A0;  //remember to connect the long leg to the positive connection
int analogvalue;
float digitalvalue, tensionvalue;  //the one coming from the potentiometer

void setup() {
  digitalWrite(LEDB, ON);  //to check if Arduino is safe
  pinMode(potpin, INPUT);
  pinMode(ledpin, OUTPUT);  //maybe it could be avoided because the analogwrite function does
it //by itself
  Serial.begin(9600);
}

void loop() {
  delay(10);
  digitalvalue = analogRead(potpin);  //the value read is included in the 0-1023 range
(standard //resolution of 10 bits)
  analogvalue = map(digitalvalue, 0, 1023, 0, 255);
  analogWrite(ledpin, analogvalue);  //the PWM range is 0-255

  tensionvalue = digitalvalue * 3.3 / 1023; //used to scale the read value to the voltage, in
//alternative still use the map function
  Serial.println(tensionvalue);
}
```

Here it is possible to change the LED brightness in an analogical way. Despite what done before, the PWM signal, sent to the LED, in charge of brightness variation, is linked to the potentiometer output value. The output value is read, scaled and sent to the LED pin. Along with that, the tension signal coming from the potentiometer is printed to the Serial Monitor.

2.1.7 eMotor Sketches

The Arduino Portenta H7 board played a crucial role in the initial phases of motor control and testing. During the bench testing of the system, the Arduino was used to generate a PWM signal, allowing us to understand the working principle of the controller. Additionally, analog devices were connected to the Arduino, and it was employed to read the output values of the controller. Being early stages of testing the battery is not used and the power comes from the power supply on the backside, once set the right tension level.

The following first 3 codes are used during the Case 1, while the last one is made for Case 2.

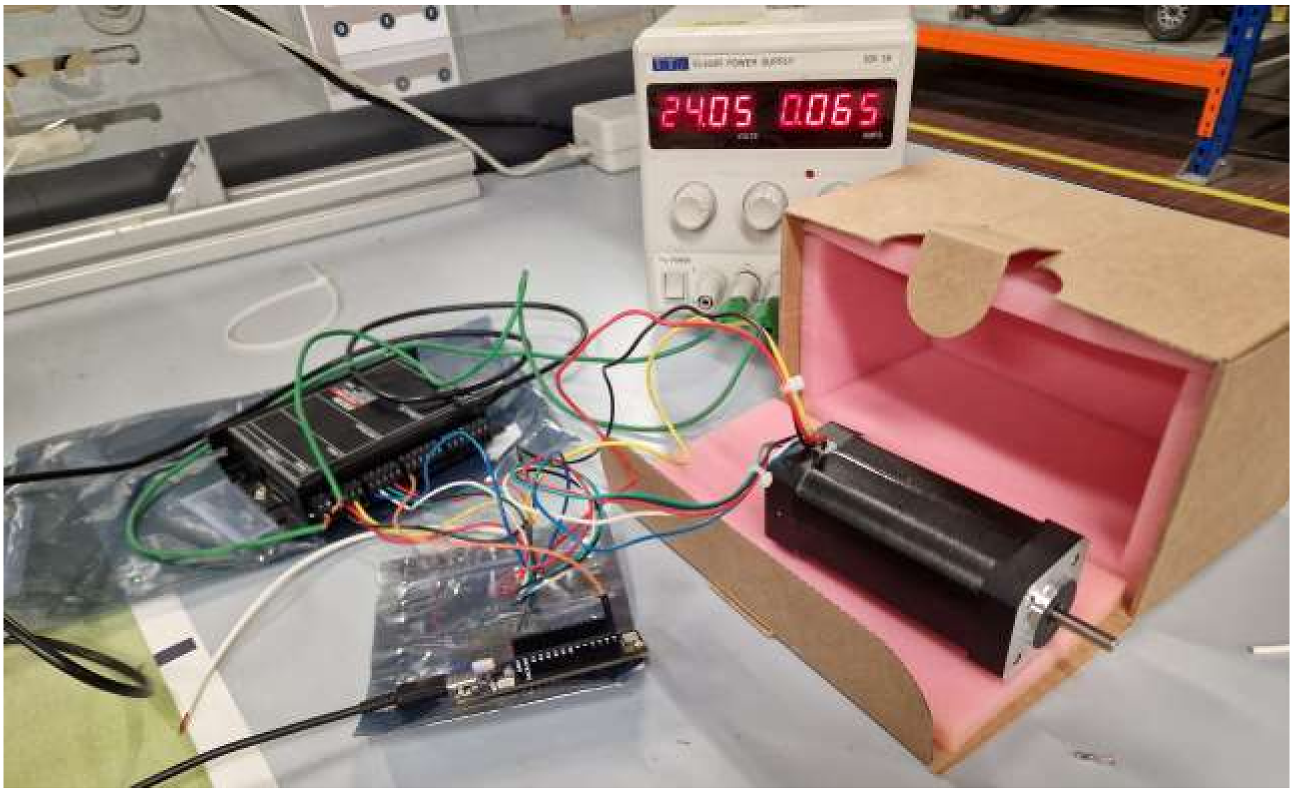


Figure 20. eMotor Bench Testing Arduino Board.

eMotor Controller Enabling Signal

```
//eMotor enabling signal - fixed speed

#define ON LOW
#define OFF HIGH

int speedpin=D5, speed=1000; //speed defined in rpm - orange one
int ctrlpin=D6;              //pin what abilitate the controller - blu one

long int clspeed;
```

```
void setup() {

  pinMode(speedpin, OUTPUT);
  pinMode(ctrlpin,OUTPUT);

  digitalWrite(LEDB, ON);       //turning on the built LED
  digitalWrite(ctrlpin, HIGH);  //motor enabling signal

}

void loop() {

  pwm=map(speed, 0, 4000, 25, 230); //PWM range merged with controller range (10%-90%)
  analogWrite(speedpin, pwm);

}
```

This Arduino code serves two purposes for motor control. First, it generates an enabling signal for the controller by setting the D6 pin to a digital HIGH state. Secondly, it sets the motor speed by means of a PWM signal. The specific thresholds for the PWM signal will be elaborated in the Experimental Tests chapter. The "map" function is utilized to convert a speed value to the corresponding PWM value.

In this case, the speed variation could be done by modifying the “speed” variable and to reupload the code.

eMotor Controller Enabling Signal with Controller Feedback

```
//eMotor fixed speed  - controller feedback signal
#define ON LOW
#define OFF HIGH

int speedpin=D5, speed=1000; //speed defined in rpm - orange one
int ctrlpin=D6;              //pin what abilitate the controller - blu one
int analogspeedpin=A0;       //pin aiming to read speed - dark orange cable
int pwm;

long int clspeed, readvalue;

void setup() {

  pinMode(speedpin, OUTPUT);
  digitalWrite(LEDB, ON);
  pinMode(ctrlpin,OUTPUT);
  digitalWrite(ctrlpin, HIGH);

  pinMode(analogspeedpin,INPUT);

  Serial.begin(9600);

}

void loop() {

  pwm=map(speed, 0, 4000, 25, 230); //PWM range merged with controller range
  analogWrite(speedpin, pwm);

  analogReadResolution(16); //now the range should be 0-65535
  readvalue=analogRead(analogspeedpin);
  clspeed=map(readvalue, 0, 65535, 0, 4000);

  Serial.println(clspeed);

}
```

This Arduino sketch is like the previous one, but with an additional feature of reading and printing a voltage level signal from the controller, which represents the instantaneous eMotor speed value detected by the controller. Like the previous sketch, the speed is a fixed parameter that can be configured during the compilation process. If any changes to the speed are desired, the sketch needs to be reuploaded to the Arduino.

External Potentiometer Regulation and Controller Feedback

```
//eMotor PWM controlled, according to a board linked external potentiometer -  controller feedbback - data printing
#define ON LOW
#define OFF HIGH

int runTime;                //time used to print data
int speedpin=D5;            //speed defined in rpm - orange one
int ctrlpin=D6;             //pin what abilitate the controller - blu one
int contr_pin=A0;           //pin aiming to read speed - dark orange cable
int pwm;                    //pwm values to be sent to the controller
int pot_pin=A2;             //potentiometer cables GND>white, +3.3V>green, signal>blu

// long int clspeed, motor_readvalue;//closed loop speed coming from the analog read value
long int pot_readvalue, contr_readvalue, contr_speed, pot_speed;  //speed coming from the analog read value and the potentiometer one

void setup() {
  pinMode(contr_pin,OUTPUT);    //set to the output the ON/OFF motor signal pin

  pinMode(speedpin, OUTPUT);    //set to output the PWM pin
  digitalWrite(LEDB, ON);       //turns on the built blue led

  pinMode(contr_pin,INPUT);     //set the analog speed signal pin as input
  pinMode(pot_pin,INPUT);       //set the potentiometer signal pin as input

  Serial.begin(9600);
}

void loop() {
  runTime=millis();

  analogReadResolution(16);   //now the range is 0-65535
  pot_readvalue=analogRead(pot_pin);  //read analogical value coming from the potentiometer
  contr_readvalue=analogRead(contr_pin); //read analogical value coming from the controller

  if (pot_readvalue <= 1000) {            //needed to smooth scattering movements of the motor, setting a low speed treshold of about 61 rpm
    digitalWrite(ctrlpin, LOW);//set the motor OFF status
  } else {
    digitalWrite(ctrlpin, HIGH);//set the motor ON status
  }

  pwm=map(pot_readvalue, 0, 65535, 26, 229); //the first interval is related to the analog potentiometer speed value, the second one is PWM range merged with controller range
  analogWrite(speedpin, pwm);

  // if (contr_readvalue>=100 && contr_readvalue<=65435) {  //needed to compensate controller
losses, more or less 0.005V
  // contr_readvalue=contr_readvalue+80;
  // }

  contr_speed=map(contr_readvalue, 0, 65535, 0, 4000); //map the controller value into speed
  pot_speed=map(pot_readvalue,0, 65535, 0, 4000); //map the potentiometer value into speed

  Serial.print("Time: ");
  Serial.print(runTime);
  Serial.print(" Potentiometer speed: ");
  Serial.print(pot_speed);
  // Serial.print(pot_readvalue);
  Serial.print(" - Controller Speed: ");
  Serial.println(contr_speed);
  // delay(50);    //useful for Putty output but introducing errors while coolterm recordings
}
```

In this case the speed is regulated by means of an external potentiometer. The regulated voltage coming from the potentiometer is processed by the board and sent to the controller in a PWM signal form. The “analogread” resolution is enhanced to its maximum. The potentiometer is fed directly from the 3V3 pin, what provides 3 volts, the green cable, the white one is the GND, while the blue is the regulated voltage signal.

The dark orange cable is a voltage level coming from the eMotor controller representing the read instantaneous speed, printed to the Serial Monitor by Arduino.

In addition to that a conditional statement is added to switch the enabling signal from ON to OFF accordin to the desired speed, it is woth nothing that 1000 is not the rpm speed value but the bit one, corresponding to more or less 61 rpm, value below what the eMotor starts moving in a scattered way.

In the code, there is a commented section where a data correction process was implemented. This was necessary because there was a slight discrepancy between the desired speed set by the potentiometer and the actual speed measured from the controller. Initially, this difference was attributed to voltage losses. However, upon further analysis, it was determined that the dynamics of the eMotor were contributing to this variation.

The print part was more detailed to have a more readable Serial, even because a process of data recording was done to analyse data in a better way.

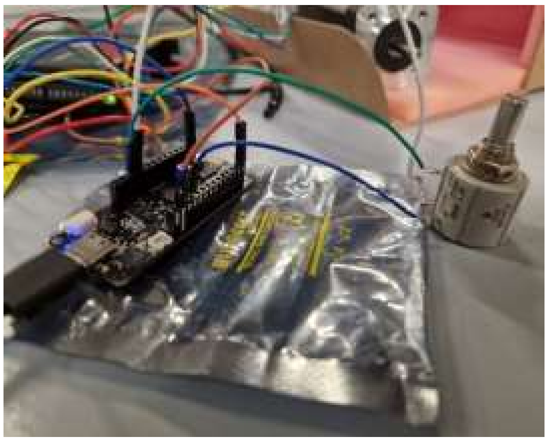


Figure 21. eMotor Analogical Signal Devices.

eMotor Controller Feedback/Potentiometer Request Signals Data Recording

```
//eMotor speed signals comparison among external potentiometer (controller directly linked)
and controller speed feedback
#define ON LOW
#define OFF HIGH

int contr_pin=A0;       //pin aiming to read speed signal coming from the controller -
burgundy cable
int pot_pin=A2;         //pin where the potentiometer is linked - dark orange cable

long int pot_readvalue, contr_readvalue, contr_speed, pot_speed, runTime;  //speed coming from
the analog read value and the potentiometer one

void setup() {

  digitalWrite(LEDB, ON);     //turns on the built blue led

  Serial.begin(9600);

}

void loop() {

  runTime=millis();
  analogReadResolution(16);   //now the range is 0-ù

  pot_readvalue=analogRead(pot_pin);  //read analogical value coming from the potentiometer
  contr_readvalue=analogRead(contr_pin); //read analogical value coming from the controller

  // if (contr_readvalue>=100 && contr_readvalue<=65435){  //needed to compensate controller
losses, more or less 0.005V
  //   contr_readvalue=contr_readvalue+150;
  // }


  pot_speed=map(pot_readvalue,0, 65535, 0, 4000); //map the potentiometer value into speed
  contr_speed=map(contr_readvalue, 0, 65535, 0, 4000); //map the controller value into speed

  // //speed values with captions
  // Serial.print("Time: ");
  // Serial.print(runTime);
  // Serial.print(" Potentiometer speed: ");
  // Serial.print(pot_speed);
  // Serial.print(" - Controller Speed: ");
  // Serial.println(contr_speed);
  // delay(50);    //useful for Putty output but creating some errors during coolterm
recording

  // //speed values no captions
  // Serial.print(runTime);
  // Serial.print(" ");
  // Serial.print(pot_speed);
  // Serial.print(" ");
  // Serial.println(contr_speed);

  // //bit values with captions
  // Serial.print("Time: ");
  // Serial.print(runTime);
  // Serial.print(" Potentiometer Bit: ");
  // Serial.print(pot_readvalue);
  // Serial.print(" - Controller Bit: ");
  // Serial.println(contr_readvalue);
  // delay(50);    //useful for Putty output but creating some errors during coolterm
recording

  //bit values no captions
  Serial.print(runTime);
  Serial.print(" ");
  Serial.print(pot_readvalue);
  Serial.print(" ");
  Serial.println(contr_readvalue);
}
```

This sketch was developed to address the issue of incomplete overlapping between the requested speed values and the actual feedback received from the controller. It reads the voltage levels from both the potentiometer and the controller's speed output, mapping them to the appropriate range.

It is worth noting that no enabling signal is produced by the Arduino board, this because the task was moved to a switch directly connected to the controller.

There is a commented section in the code that was initially intended to compensate for voltage losses but was later discovered to be a mistake and left as a comment, as already explained in the previous section.

The last part of the sketch includes various printing options, some of which are commented out. These print statements were used to record the data sent to the Serial Monitor. Depending on the specific requirements, either a more detailed caption or just raw data could be selected for printing.

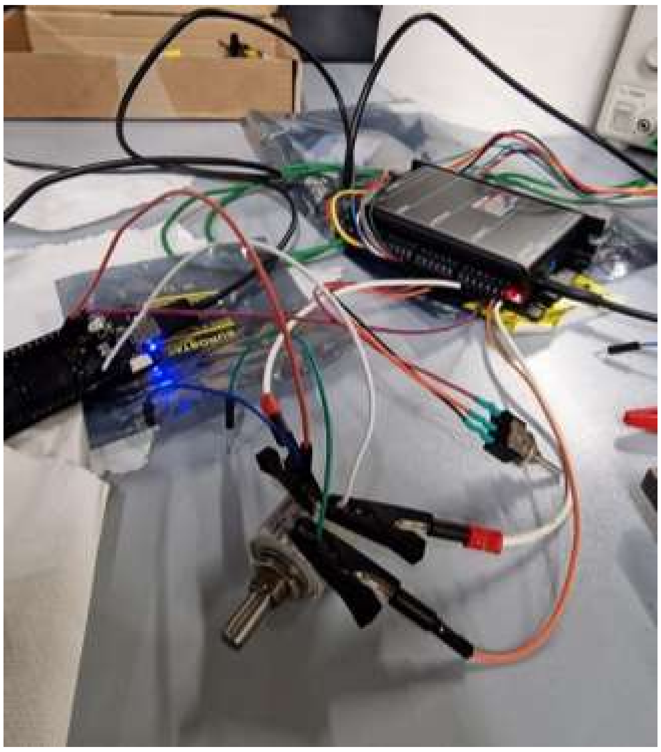


Figure 22. Devices for Speed Signals Comparison.

2.2. Maxon Escon Controller

2.2.1 Introduction

To ensure accurate coordination of the phases in an eMotor, a dedicated device called an eMotor controller is required. The primary function of the controller is to synchronize the phases and adjust the speed or torque of the motor based on a reference signal provided by an external source. The control signal can be of different types, such as PWM (Pulse Width Modulation) or analog voltage levels.

The eMotor controller directly influences the phases of the eMotor to minimize any discrepancies between the input value and the motor's response. This control process can be achieved through closed-loop control, facilitated by the built-in Hall sensor. The controller continuously monitors the motor's performance and adjusts the phase coordination accordingly to maintain a closer match with the desired reference signal.

The e-motor used is a three-phase brushless DC one, with a power of 480 Watt, 24V voltage alimentation and a peak current of 20 Amps. The controller able to satisfy these requirements is the Maxon Motor Escon 50/5.

This controller offers a variety of connections to better manage control signals or data coming from the motor, like the adsorbed current, the effective shaft speed etc. While in input could accept turning on/off signals or sudden motor stop.

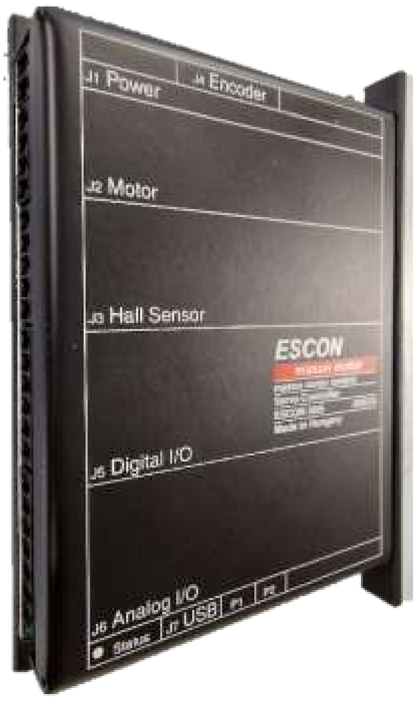


Figure 23. Controller connectors overview, with detached plug.

2.2.2 Characteristics

The ESCON 50/5 is a small-sized, powerful 4-quadrant PWM servo controller for the highly efficient control of permanent magnet-activated brushed DC motors or brushless EC motors up to approximately 250 Watts, peaking at about 500 Watts.

How to command

The controller needs a proper reference to command the motor, this reference could be provided in different ways:

- Fixed Value, a value is chosen during the setup and the controller will follow that one, useful only for bench testing or easy to switch off applications;
- PWM Signal, this signal could be processed and, according to the duty cycle, the controller will regulate the speed. The duty cycle limits are 10% and 90%, so the input PWM must be tuned according to that range;
- Analog value, a voltage value could be used for the reference, this could be a fixed one or an adjustable one, by means, for instance, of a potentiometer;
- Lastly the reference could be set, and varied, moving the built potentiometer, for sure not the smartest way of doing it, because of difficulty in accessing to them, but in some cases could be an effective solution.

Connections

The controller is equipped with a series of plugs, Figure 24 in which is possible to connect the proper wires. On the top of the controller the type of connection managed by the correspondent plug is printed, while on each plug port is labelled by a printed number, helpful to deal with a large number of cables.

The connectors are plastic elements in which the wires are plugged and tightened with a small screw. The cables don't require a specific terminal, even a stripped cable fits well; to better have no short circuit, it might be useful to have the cable insulating layer directly to the plug. The pitch is 3.5mm.

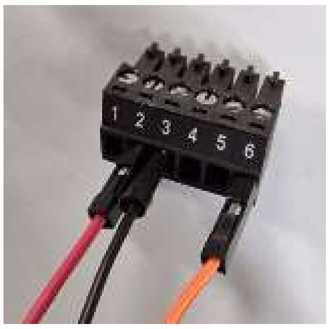


Figure 24. Controller Connector.

A highly beneficial feature of the controller is its ability to detach each plug. This allows for the convenience of leaving the plugs connected to their respective devices, saving time during future connections. For example, the motor and Hall Sensor plugs, which consist of 8 cables, can be detached from the controller, and stored together with the devices they are connected to. When needed again, the connectors can simply be plugged back into the controller, eliminating the need for time-consuming reconnections. This feature enhances the overall efficiency and convenience of the system.

In addition to time saving, the plugs detaching feature improves the wires management, allowing to work in a more comfortable position, closer to the device what should be connected, avoiding interference with the already linked cables too.

This controller is able to deal both with digital than analogical signals, in input or in output. The ports in charge of doing so, are denoted with *Digital I/O* and *Analog I/O*. The digital input signals could be PWM or just digital HIGH and LOW, while the analogical one is just a voltage level.

Following, a focus on each connector is denoted. [2]

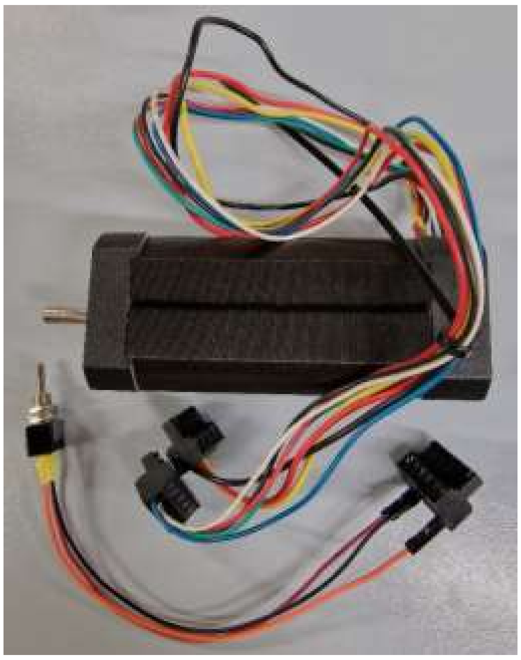


Figure 25. Devices with its own plugs.

Power Supply J1

This plug is used for the power source connection, it could be linked to a power supply unit, a battery or whatever could provide electric power. The controller is able to deal with voltage ranging from 10V to 50V, continue current; the voltage supply source must be chosen according to the needs of the motor to be controlled, so in the application case a 24V is the chosen one.

The port numbered with one must be connected to the GND, the other labelled with 2 to the positive wire.

Motor J2

This plug serves the purpose of interfacing with the motor by connecting its various windings, depending on the type of motor being used. The controller is capable of accommodating both brushed and brushless DC motors, as well as EC motors (Electronically Commutated), with or without an Encoder or a Hall Sensor. It is essential to choose the appropriate connection based on the specific motor configuration to ensure proper functionality and compatibility with the controller.[3]

Hall Sensor J3

This connector aims to deal with the motor Hall Sensor, if present. The connector counts on five pins, the first three are the inputs, while the 4th is the +5VDC source, needed by the sensor, and lastly, the 5th is the ground.

Generally, if a Hall Sensor is built into an e-motor the ground cable is shared among the motor and the sensor, so the 4th motor connector pin is left unwired.

[2] More specs are available in the [10], only the relevant is discussed.
[3] Have a look at the *Motor connection* paragraph for the complete wirings.

Encoder J4

In case of Hall Sensor absence or in addition to it, an encoder could be plugged, this one does not have a connector like the other ones but a specific socket, site on the top of the controller. For the connection a specific encoder plug is needed, this is the DIN 41651, which counts on 10 poles.[4]

Digital I/Os J5

The controller provides a 6-pin connector used for digital signals input/output, this can work with voltage levels ranging from 0 to 36 VDC, negative voltage is managed too.[5]

The controller is so composed:

- the first two pins of the plug are specifically designed to read digital signals. The first pin has the capability to handle PWM signals, which can be used to modify the motor's speed or torque in a manner similar to an analog signal. The second pin is dedicated to digital signals, with a logic zero value below 1V and a logic one value above 2.4V. These pins provide the necessary interface for controlling and communicating with the motor using digital signals;
- the pins 3 and 4 behave in the same way, they're both able to work as input or output, the voltage range is still the same of the previous two pins;
- the pin number 5 is a ground signal, used for the negative side of the digital circuit. The controller does not share the same ground with all the connector plugs, each connection has its own; due to that is possible to connect devices working in a different way, e.g., a switch will be connected to the digital port, while a potentiometer to the analog one, having so separate circuits;
- the last pin, the number 6, provides an output voltage level of 5V, used for digital devices feeding.

Analog I/Os J6

The last controller connector is a 7 pins plug used to deal with analogical signals. The controller translates the analogical value into a digital one, for processing, thanks to an included 12-bit ADC. The input pins are used in couples, on the contrary of the analog outputs what work individually. Each pin (or pair) deals with a specific voltage range.[6]

The controller pins are so distinguished:

- Plug's pins 1, 2, 3, and 4 are specifically designated for analog input. These pins are configured to handle a differential voltage signal ranging from -10V to +10V. Each pair of pins (1 and 2, as well as 3 and 4) manages the positive and negative voltage differentials of the signal, respectively. In cases where a device is unable to provide a negative differential signal, it is necessary to connect pin 2 to the circuit's ground (GND) to ensure proper functioning of the controller. This configuration was demonstrated in Case 2 – Potentiometer Control, where pin 2 was connected to the ground of the potentiometer circuit. It is important to note that the second input pair (pins 3 and 4) should not be used independently, as they are meant to function together in handling the analog signals.

[4] For more information check the [10], pages 16-17.
[5] The precise technical data are available in the [10], chapter 3.3.5.
[6] The precise technical data are available in the [10], chapter 3.3.6.

- The pins 5 and 6, instead, are used in an independent way and produce an analogical output value, ranging from -4VDC to 4VDC. It is worth nothing again that the GND is local and not the same for the whole controller, this means that a proper GND connection must be done. The controller moves the internal digital values, to the analogical ones, passing through a 12-bit DAC. Those pins could be set on a certain voltage level to feed an external device, according to its request.
- The pin number 7 is the analog signals ground.

Micro USB J7

The controller features a micro-USB type A port, which serves as the interface between the controller and a PC during the initial setup process. Once the controller has been properly configured, it can be disconnected from the PC and operate offline.

Potentiometers

On the underside of the controller, there are two 240° linear potentiometers. These potentiometers can be assigned specific functions during the controller setup phase based on the requirements of the project. Typically, they are used to regulate an offline offset. For example, they can be utilized to add a speed offset without the need for a PC setting phase, resulting in significant time savings.

Status LED

The controller features a helpful status LED that indicates its operational status (enabled or disabled) and provides error indications through different blinking patterns. While interpreting the blinking patterns may not be the most efficient method, it serves as a convenient way to quickly check the error status when the controller is installed in the driveline and USB connection access may be limited.

The LED has different colours and blinking patterns. A solid green light indicates that everything is properly configured, and the controller is free from errors. A green blinking light signifies controller disabling, such as when a stop signal is received. On the other hand, a red blinking light, with varying periods, indicates different types of errors, including PWM signal out of range, thermal overload, overvoltage, undervoltage, and more.[7]

Operating Modes

The controller deals with the motor achieving two different targets, the speed control and the current control, in closed or open loop. It is worth noting the possibility of reverse energy flow management, very useful for applications in which the motor works as a brake too; according to that, the power source can accept this power.

Here a recap of the operating modes is listed:

- Speed Control – Closed Loop, this could be performed thanks to the feedback coming from an encoder, a DC Tacho, a Hall sensor or exploiting the BEMF[8];
- Speed Control – Open Loop: The controller offers the option for static or dynamic adaptive IxR compensation in speed control. The IxR compensation calculates the motor voltage based on the speed constant, which needs to be configured during the setup phase, as well as the motor current and winding resistance, which can be configured at the beginning or automatically determined during the tuning phase. The adaptive IxR compensation aims to compensate for dynamic load variations without affecting the motor speed or at least

[7] It is possible to check the complete LED interpretation table in the [10], Table 3-22, page 27.
[8] Back Electro-Motive Force.

minimizing the impact. It achieves this by adjusting the motor voltage to account for changes in motor temperature, as variations in temperature can affect the winding resistance, following the thermal resistance coefficient of the copper.[9]

- Current Control (Torque Control), the current control is performed comparing the actual motor current (torque) with the applied set value. In case of deviation, the motor current is dynamically readjusted.

2.2.3 Escon Studio

The configuration and interfacing of the controller with the eMotor is performed using the Escon Studio Software, developed by Maxon. The controller is connected to the software via the micro-USB port, allowing for convenient setup and configuration.

Getting Started

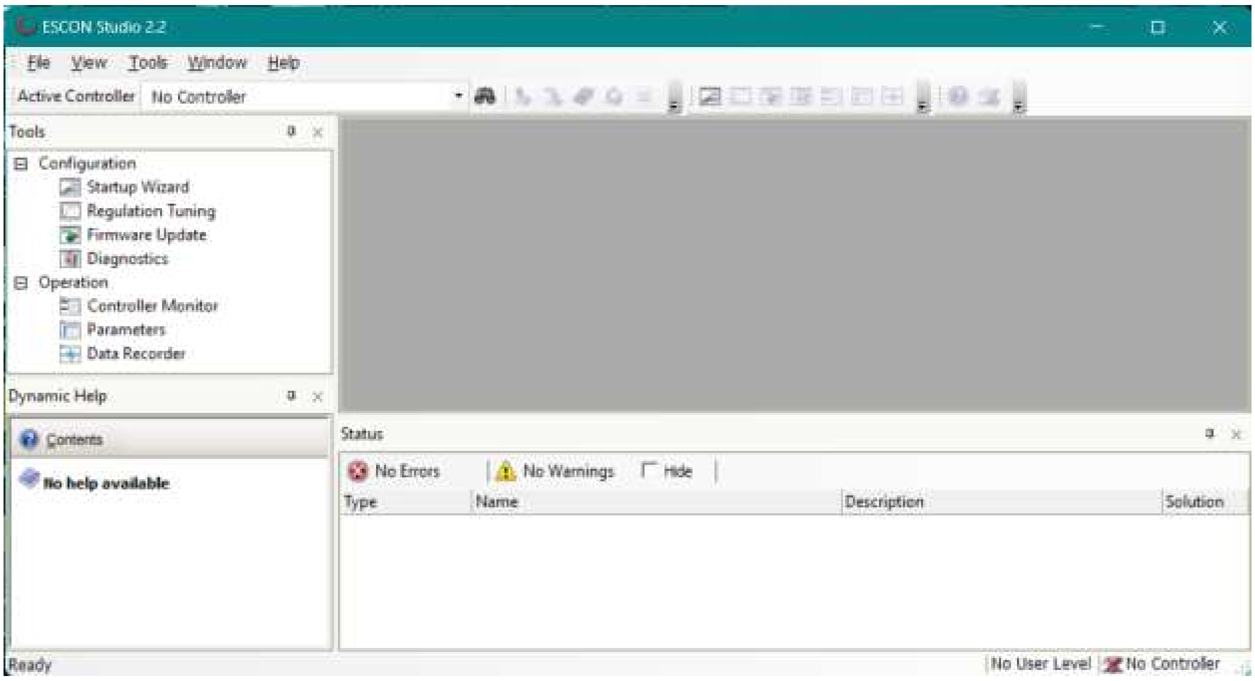


Figure 26. Escon Studio Interface.

The software interface is composed by an upper section, where, apart from the standard *File, View, Tools, Window* and *Help* drop down menu, is possible to appreciate the Active Controller tab, Figure 27. Here is possible to chose among the connected controller in case of multiple connections; since the software features a monitoring section, it could be used only to check the eMotor status. On the side a binocular button is used for detecting the connected ones.

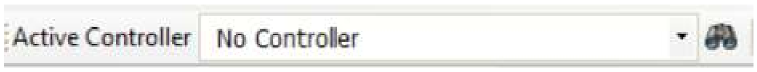


Figure 27. Active Controller Tab.

Moving to the following items, them are just a repetition on what is on the left side area and are tools useful for the controller configuration and management.

[9] More information here [8].

The lower part is divided into two main area, the left one is the *Dynamic Help,* where some advice links appear, according to what the user is doing/tuning/monitoring; on the right side the occurring errors are shown by the time they appear.

Startup Wizard

The best way to start the motor configuration is exploiting the Startup Wizard. A great feature included is the possibility to use a virtual controller among a list, tuning the selected model and uploading the parameters later. The procedure is a sequence of steps in which the motor parameters must be added; in case of parameters lack, thanks to the Regulation Tuning, they will be auto detected and added in the right place. By the steps, it is possible to add the maximum speed, current, the presence of what kind of sensor and the operating mode selection.

Following, the configuration for each port will be determined, some of the most relevant will be listed:

- The controller requires an enabling signal to activate its operation. This signal can be a digital input sourced from various devices such as a button, a switch, or a control board. The enabling signal can also be utilized to determine the rotation direction of the motor. For instance, a three-position switch can be connected, as shown in
- Figure 36. Case 1 Wirings.
- Case 2, where the position of the switch determines whether the motor rotates clockwise (CW) or counterclockwise (CCW);
- Set Value, at this point must be denoted how the controller reference is passed. The first thing to do is to select from the drop-down menu, what kind of signal will be implied, an analogical, a digital PWM, a fixed one etc. For each one, a series of options will be shown, to be tuned according to the request;
- Offset, here is possible to add an offset, it could be fixed or not. If a variable offset is chosen it could follow an external analogical signal or could be set according to the controller-built potentiometers.

At the conclusion of the configuration process, a summary of each port's assigned task is provided. Additionally, there is an option to directly access the wiring overview, which presents a printable recap of the port connections and their respective tasks. This overview can be referred to later for reference and verification purposes.

Tools

Regulation Tuning

After completing the Startup wizard, the tuning tool can be utilized to further optimize the controller's parameters through plant identification. The recommended approach is to use the auto-tuning feature, which covers the most common load cases. The system response is displayed at the end of the process and can be fine-tuned manually if desired.

In cases where specific conditions or requirements need to be addressed, the Expert tuning feature can be employed. This allows for manual adjustment of parameters and is particularly useful when working with low-resolution speed sensors, unbalanced or variable frictions, or when accounting for factors such as wear.

Firmware Update

Since the controller firmware is continuously developed by the manufacturer, sometimes it needs an update. This is possible to be done by means of this tool.

Diagnostics

The diagnostic tool provided by the Escon Studio Software is highly effective in identifying and resolving issues that may arise during the commissioning of the controller. It performs comprehensive analyses of the controller, motor, connected sensors, and built-in speed sensors to detect any missing or incorrect connections, defective components, or incorrect parameter settings.

Based on the diagnostic results, the tool offers appropriate actions and recommendations to address the identified problems. It may suggest specific tests to be performed to ensure the proper functioning of the system and to rectify any issues that may have been detected. This powerful diagnostic capability greatly assists in troubleshooting and ensuring optimal performance of the controller and associated components.

Controller Monitor

Once the system is properly configured, the Escon Studio Software provides a comprehensive overview of the input/output signals, set parameters, errors, controller status, and plant status. This overview allows users to monitor and analyze the system's performance and make real-time adjustments to certain values in the "Controller" section. By clicking on the input port, users can modify the signal value thresholds, bypassing the need to go through the Startup Wizard.

In the "Controller" tab, a detailed schematic of the controller parameters is displayed, providing users with the ability to modify specific parameters in real-time. Finally, the "Properties" tab offers a complete summary of all the selected values and configurations, providing a convenient recap of the system setup.

Overall, the Escon Studio Software offers a user-friendly interface that allows for efficient monitoring, adjustment, and management of the controller's settings and performance.

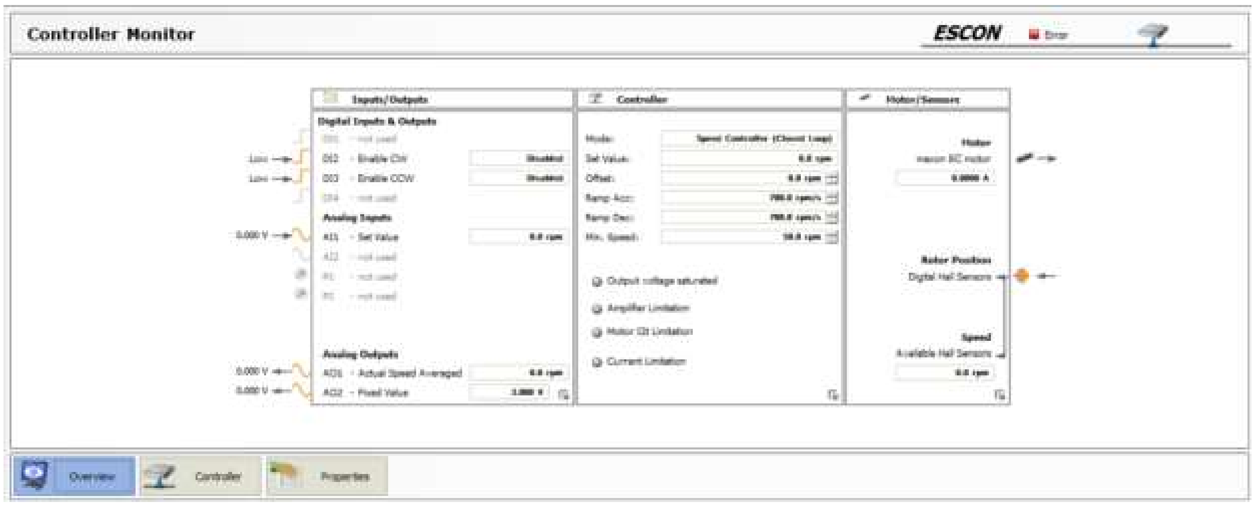

Controller Monitor
Inputs/Outputs
Digital Inputs & Outputs
DI1 - not used
DI2 - Enable CW: Disabled
DI3 - Enable CCW: Disabled
DI4 - not used
Analog Inputs
AI1 - Set Value: 0.0 rpm
AI2 - not used
Analog Outputs
AO1 - Actual Speed Averaged: 0.0 rpm
AO2 - Fixed Value: 3.000 V
Controller
Mode: Speed Controller (Closed Loop)
Set Value: 0.0 rpm
Offset: 0.0 rpm
Ramp Acc: 700.0 rpm/s
Ramp Dec: 700.0 rpm/s
Min. Speed: 50.0 rpm
Output voltage saturated
Amplifier Limitation
Motor I2t Limitation
Current Limitation
Motor/Sensors
Motor: maxon EC motor 0.0000 A
Rotor Position: Digital Hall Sensors
Speed: Available Hall Sensors 0.0 rpm

Figure 28. Controller Monitor Overview.

Data Record

This section provides a powerful data recording feature that allows users to capture and analyze up to 4 different signals over time. The plotting capabilities are highly flexible, allowing users to adjust the plot settings such as axis scales, trigger options, and time representation (e.g., samples and sampling time). Users can start the recording based on a trigger condition, ensuring precise data capture.

The recorded sessions can be exported as images or files, providing users with the ability to save and analyse the data offline. Additionally, users can upload previously recorded sessions for further analysis within the software.

This data recording feature enhances the overall functionality of the software by enabling users to capture and analyse critical signals, monitor system behaviour, and make informed decisions based on the recorded data.

Parameters Download/Upload

To enhance convenience and efficiency during testing, the controller offers a valuable feature that allows users to save and re-upload eMotor parameters and control strategies. This feature proves particularly useful during bench testing, where various configurations are tested to determine the optimal setup. With the ability to save and quickly re-upload these parameters, users can easily switch between different configurations and control signal sources, along with any other necessary input setups.

The Upload/Download buttons, located on the second row above the monitor, next to the drop-down menu, enable users to store and retrieve these configurations effortlessly. This functionality eliminates the need for repetitive parameter adjustments and control strategy reconfiguration, allowing users to swiftly transition between different test scenarios and evaluate the performance of various setups.

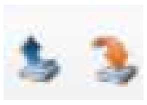


Figure 29. Upload/Download parameters buttons.

The blue arrowed one is the *Upload Parameters*, even if it seems to be counterintuitive, it is used to save an ".edc" configuration file, from the controller set-up to the PC; the other button is instead the *Download Parameters*, what takes a set-up file and upload it to the controller.

2.2.4 Wirings

In this paragraph, the recommended cable connections for the two types of controllable motors are illustrated, along with the corresponding wiring configurations for different application scenarios. It provides guidance on how to establish the proper connections to ensure optimal motor control and performance. By following these wiring guidelines, users can effectively interface the controller with the specific motor type they are using and adapt it to their desired application requirements.

Motor connection

According to the previous paragraphs, the controller is able to deal with a large combination of motor and sensors, for each combination a different wiring must be performed.

Brushed DC Motor – No Sensors

In this case the two windings are linked to the pins 1 and 2, positive on the 1 and negative on the 2, the pin number 3 is not connected, while the 4th pin manages the motor ground/shield cable. The controller can measure the back EMF coming from the motor windings, in this way, even without any sensor, is still possible to obtain a closed-loop control.

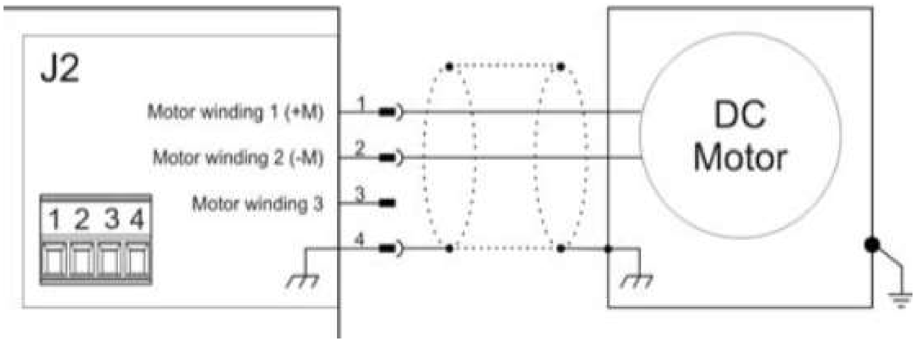


Figure 30. DC motor - No sensors wiring.

Brushed DC Motor – Encoder

The motor connection followed in this case is like the previous one, what differs is the presence of the encoder. This does not change anything apart from the dedicated connection of the sensor to the relative J4 plug.

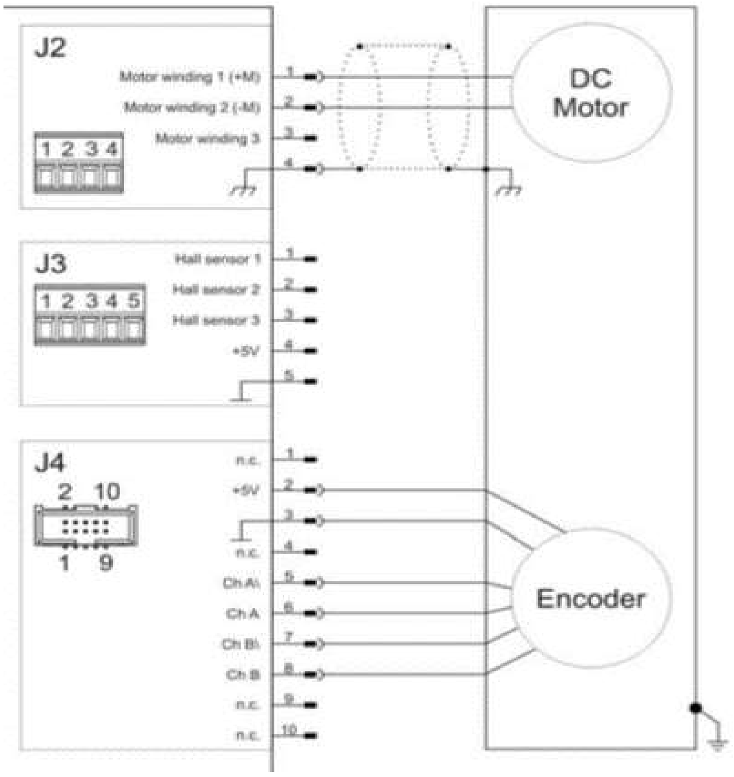


Figure 31. DC Motor - Encoder wiring.

Brushed DC Motor – DC Tachometer

As an alternative to using an encoder, a DC tachometer can be utilized for feedback control, depending on the available options for motor control. The DC tachometer needs to be connected to an analog input pair pin, such as pin 1-2 or 3-4, and to the ground pin, which is pin 7 if available. During the controller setup phase using the Escon Studio Software, the J5 Analog I/O connector plug will be properly configured to accommodate the DC tachometer. It is important to note that the motor wiring remains unchanged in this configuration.

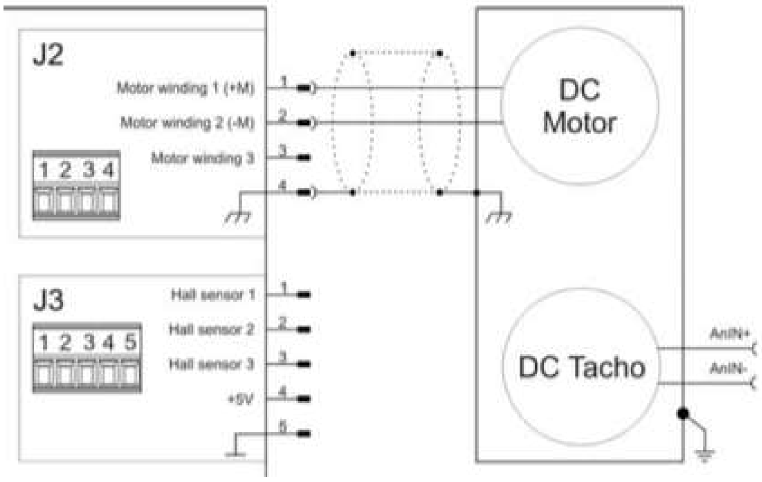


Figure 32. Brushed DC Moto – DC Tachometer wiring.

Brushless DC Motor – Hall Sensor

When utilizing a Brushless DC Motor (EC motor) with an embedded Hall sensor, the motor windings are connected in sequential order to the first three pins of the J2 connector. The Hall sensors are typically integrated within the motor's stator, eliminating the need for external sensor placement. In this configuration, the Hall Sensor shares the same ground cable as the motor windings, and pin number 4 of the motor connector is left unused. For the Hall Sensor, there are three signal wires that correspond to the three sensors within it. These signal wires should be connected in accordance with the sequence of the motor windings. Additionally, the Hall Sensor requires a 5 Volts supply, which is provided through pin number 4 of the J3 Hall Sensor connector.

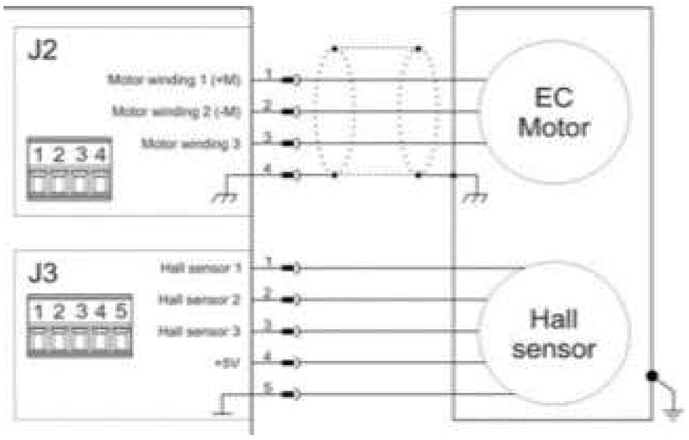


Figure 33. Brushless DC motor – Hall Sensor wiring.

Brushless DC Motor – Hall Sensor and Encoder

In the configuration involving a Brushed DC Motor with an encoder, or even with a brushless motor equipped with an encoder, the encoder can be easily connected to the J4 Encoder Connector. The wiring for this scenario is similar to the previous one, with the addition of the encoder cable. The encoder's signals and power supply are integrated into the J4 Encoder Connector, simplifying the connection process.

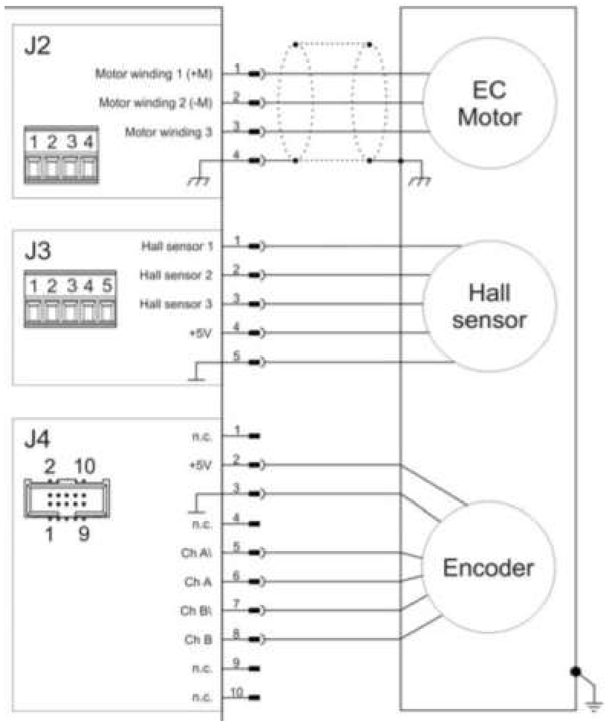


Figure 34. Brushless DC motor – Hall Sensor and Encoder wiring.

Application Wirings

The eMotor control was performed in different ways, both for a sake of study all the offered features then for the search of the best control method.

Even if the digital and analog ports connections are different among the cases, the eMotor, Hall sensor and power source connections are still the same, as in the following picture, the numbers at the end of the arrows represent the plug printed number, the wires colours follow the existing ones.

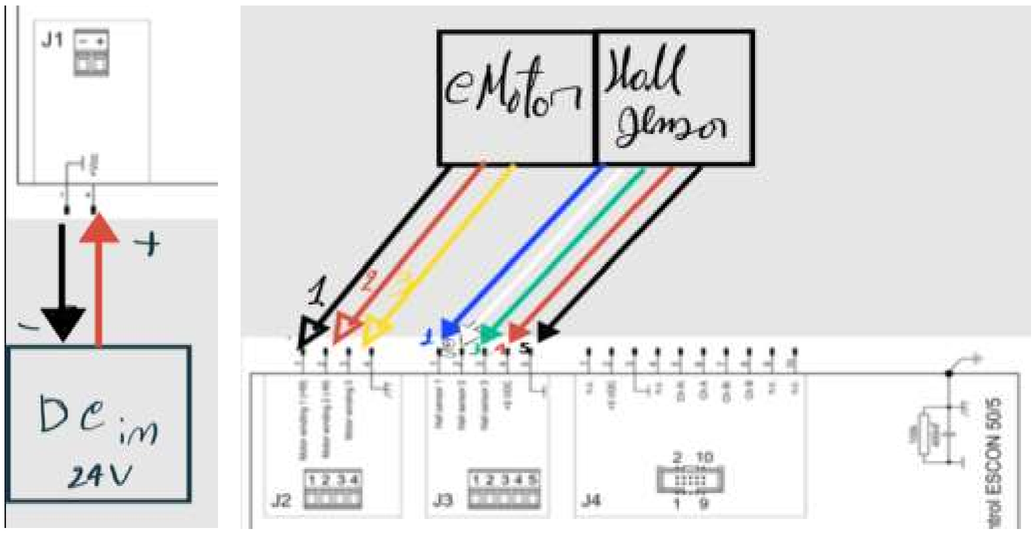


Figure 35. eMotor/Battery - Controller Wirings.

According to the command signal and used external devices, the connections changes and are grouped as follows[10]:

Case 1 – Static Test

In the given scenario, an Arduino board is being used to control the speed of a device. The speed signal is sent from the Arduino to the controller using a PWM (Pulse Width Modulation) signal. To generate the PWM signal, an Arduino pin capable of PWM generation is selected, which in this case is pin D5.

The PWM signal's duty cycle is adjusted based on an analog signal obtained from a potentiometer. The potentiometer provides an analog value that is decoded by the Arduino board and used to determine the appropriate duty cycle for the PWM signal. By changing the duty cycle, the speed is regulated. The Arduino board is also responsible for generating the enable signal, which is a digital HIGH signal.

Both the Arduino board and the controller are connected to a PC via separate USB cables. The Arduino board requires a USB Type-C connector, while the controller uses a Micro-USB connector. These USB connections allow communication between the Arduino board, the controller, and the PC. The PC can send commands or receive data from both the Arduino board and the controller, facilitating control and monitoring of the system.

The numbers near the arrows represent the printed pin/port numbers, for the sake of clarity and easy reference in the diagram.

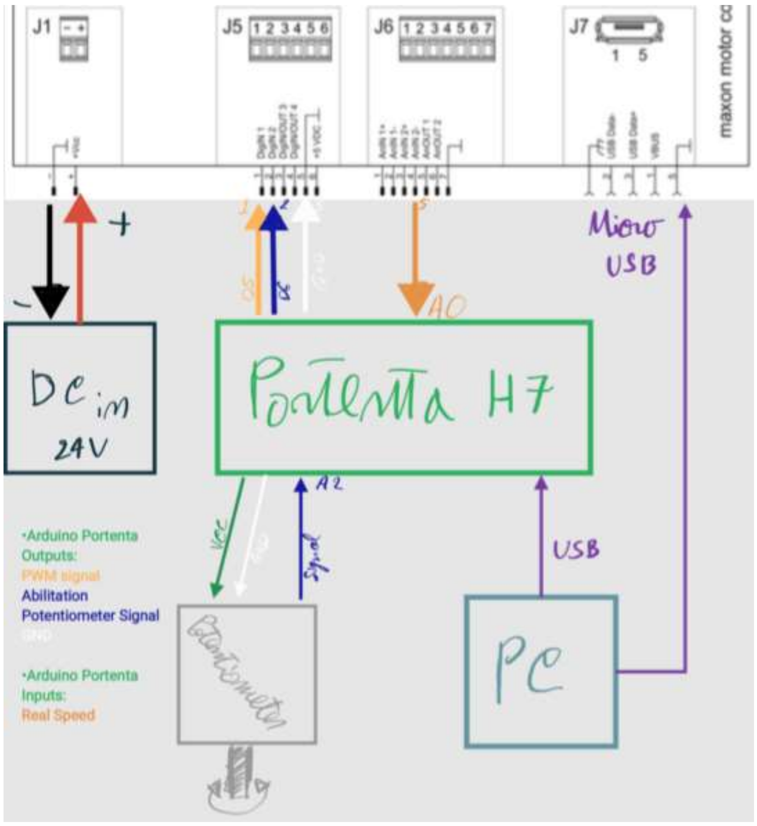


Figure 36. Case 1 Wirings.

[10] The complete explanation of the tests performed is done in the Experimental Tests chapter.

Case 2 – Potentiometer Control

In this updated scenario, the control signal for the speed of the eMotor is analog, eliminating the need for an Arduino board to convert the signal. The potentiometer is directly connected to the controller, and it is powered by the +5V J6 plug port. The white cable serves as the ground (GND) connection for the potentiometer. Port 5 on the controller is responsible for transmitting the real-time speed of the eMotor to the Portenta board.

The system incorporates a 3-position switch that is used to set the enabling signal. The switch is powered by the +5V J5 port. Depending on the desired direction of rotation and speed, the two positions of the switch are connected to ports 2 and 3 on the controller.

Although the control signal is no longer processed by the Arduino board, it still plays a role in the system. The Arduino board is used for comparing and visualizing the speed signals through the Serial Monitor feature.

The PC remains connected to the system for debugging purposes, allowing communication with the Arduino board and the controller. This enables real-time monitoring, data analysis, and adjustment of the system parameters using the PC's interface.

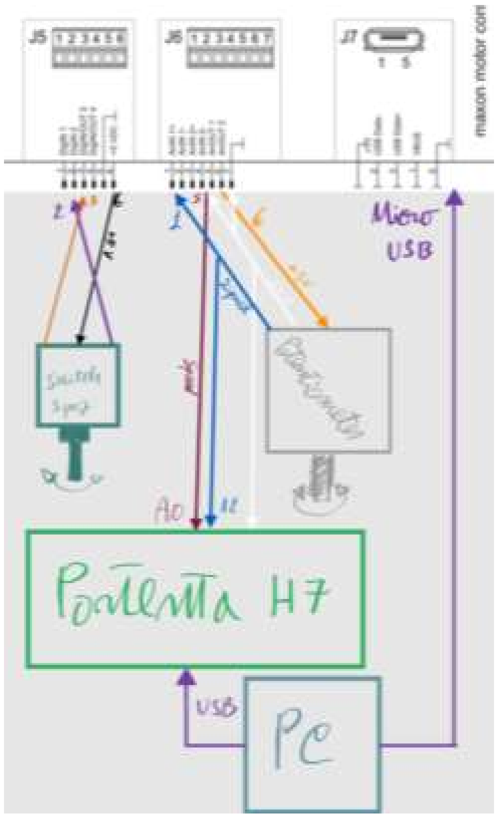


Figure 37. Case 2 Wirings.

Case 3 – Wireless Control

This scenario is similar to the first one, as the speed signal is once again a digital PWM. However, wireless devices are implied: a receiver (RX) and a transmitter (TX).

From a wiring perspective, there are two wires connecting the RX device to the controller. The black wire is used to carry the signal from the RX device to the controller, while the white wire serves as the ground (GND) connection, ensuring a common reference voltage.

The switch from the previous case is still present and performs the same tasks. It is used to set the enabling signal, and the two positions of the switch are connected to the respective ports on the controller to determine the desired direction of rotation and speed.

This wireless transmission allows for more flexibility in the system's setup and eliminates the constraints of wired connections.

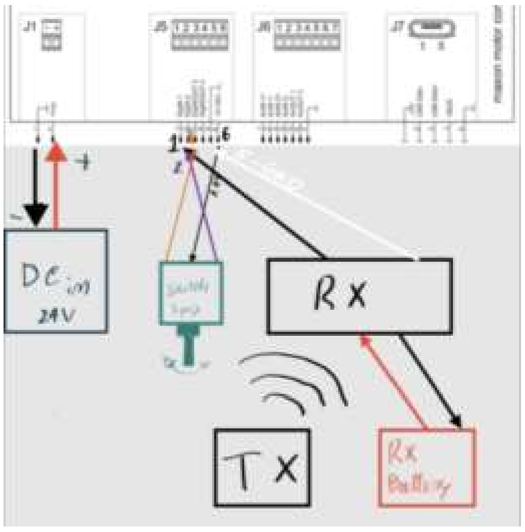


Figure 38. Case 3 Wirings.

2.3. Speedgoat

2.3.1 Introduction

The Speedgoat system is a powerful and versatile real-time target machine that has been employed in the thesis work to enhance vehicle control. By seamlessly integrating hardware and software components, the Speedgoat system offers a highly efficient development and prototyping environment for real-time control applications, simulations, and hardware-in-the-loop (HIL) testing.

Figure 39. Speedgoat Baseline.

One of the key strengths of the Speedgoat system lies in its extensive range of I/O modules, which provide exceptional flexibility for interfacing with external devices, sensors, actuators, and complex control systems. In the present application, the IO 397 Configurable I/O Module and Simulink-Programmable FPGA I/O Module have been employed.

The I/O 397 module is designed with a variety of spring-loaded pins that facilitate easy cable connections, signal routing, and even power supply if needed. It consists of two parts, with one dedicated to analog signals and the other focused on digital signals. The module features 4 16-bit DACs, 4 16-bit ADCs, and 17 digital pins, each serving different tasks based on specific requirements.[11]

The pin mapping within the Speedgoat system aligns with different operating modes, with analog pins functioning consistently across modes while digital pins exhibit variations. These operating modes include Communication, Hardware in The Loop (HIL), and Rapid Control Prototyping (RCP). Each mode offers distinct features and capabilities. For instance, in the HIL operating mode, 6 digital pins are assigned to signal capturing, 3 serve as generic I/O pins, and one is dedicated to interrupt functionality.

The system places a special emphasis on reading/writing PWM signals and managing analog signals, as these play a crucial role in various applications. PWM signals are widely used both in the stock configuration, for controlling servo motors, and in the electrified configuration, where they govern the eMotor and steering servo.

Along with the cited interface, another interface is present: the I/O 691 module. It provides an intelligent CAN interface with two channels that support flexible data-rate CAN (CAN FD) and high-speed CAN (CAN HS). This works by means of Simulink blocks as in the previous case.

With its comprehensive set of features and adaptable configuration options, the Speedgoat system proves to be an invaluable asset for achieving enhanced vehicle control and realizing advanced control strategies in real-time applications.

The study of this device was done by an in depth understanding of the provided examples and some constructed models.

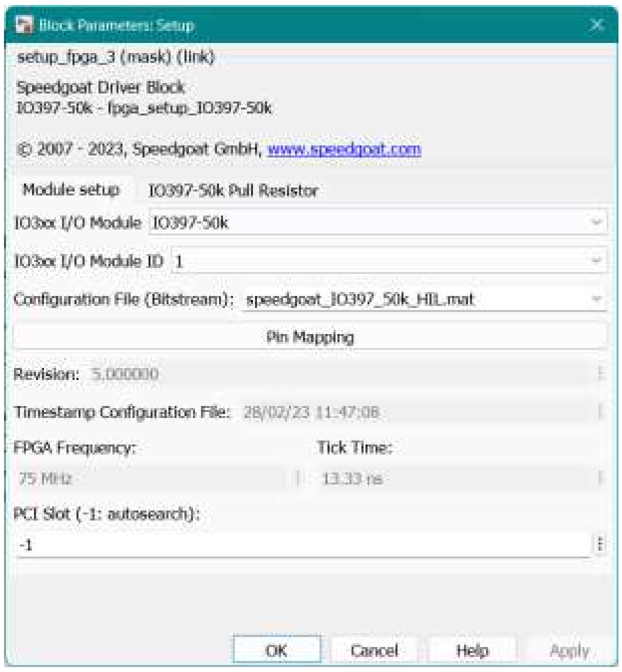
2.3.2 Simulink interface

Speedgoat is a hardware device that works in conjunction with Simulink Real-Time models. Simulink Real-Time models are like regular Simulink models, in addition to some blocks specific for the device, enabling the communication between it and the model. These blocks are available in the Speedgoat Configuration Package.

To use Speedgoat, the device needs to be connected using the Real-Time Explorer. This tool allows you to detect the IP address of the machine, update the system, reboot the device, and load models to be executed offline without the need for a PC.

The setup block is an essential component that enables the interaction with the Speedgoat device. Within this block, you can select the working mode, choose the specific module to be used, configure the system using a configuration file, and adjust other relevant parameters. The "Pin Mapping" tab within the setup block provides a helpful overview of each pin's task based on the selected configuration file. This feature assists in quickly wiring the system for the specific application case.

[11] A detailed specification is available here [16].



Block Parameters: Setup

setup_fpga_3 (mask) (link)

Speedgoat Driver Block
IO397-50k - fpga_setup_IO397-50k

© 2007 - 2023, Speedgoat GmbH, www.speedgoat.com

Module setup   IO397-50k Pull Resistor

IO3xx I/O Module: IO397-50k

IO3xx I/O Module ID: 1

Configuration File (Bitstream): speedgoat_IO397_50k_HIL.mat

Pin Mapping

Revision: 5.000000

Timestamp Configuration File: 28/02/23 11:47:08

FPGA Frequency: 75 MHz

Tick Time: 13.33 ns

PCI Slot (-1: autosearch): -1

Figure 40. Speedgoat Setup Block Parameters.

In addition to the setup block, this application involves three main blocks: Analog Input, PWM Capture, and PWM Generation. These blocks provide functionality for reading analog signals and generating or capturing PWM signals.

The Analog Input block is used to read analog signals. It can be configured to adjust the acquisition voltage level, allowing for more precise readings. For example, you can set it to a 0-5V range instead of a -10 to 10V range, while still maintaining a 16-bit resolution. The block outputs up to four analog signal voltages.

The PWM Generation block is used to generate PWM signals. It can generate up to three different signals, labelled alphabetically. In the following picture, the block is set to receive PWM duty cycles, but it can also handle ON and OFF times for asymmetric signals. The signals can be independent of each other or synchronized. In the synchronized mode, when signal A is ON, signal B is OFF, and vice versa. In this case, only one set of parameters needs to be configured. However, if the signals are independent, parameters must be selected for each signal. The symmetric working mode can be used in control applications to provide redundancy for safety purposes.

The PWM Capture block is the dual version of the PWM Generation block. It only works when the Hardware-in-the-Loop (HIL) configuration is selected, whereas the PWM Generation block works in the Rapid Control Prototyping (RCP) case. The PWM Capture block has several outputs, including the duty cycle based on 8 or 16 bits, depending on the desired accuracy. Other outputs include the high signal switch time, the ON duration, and so on. The configuration allows for reading up to six different PWM signals.[12]

[12] Each block accounts for a well detailed Help as the standard Simulink blocks.

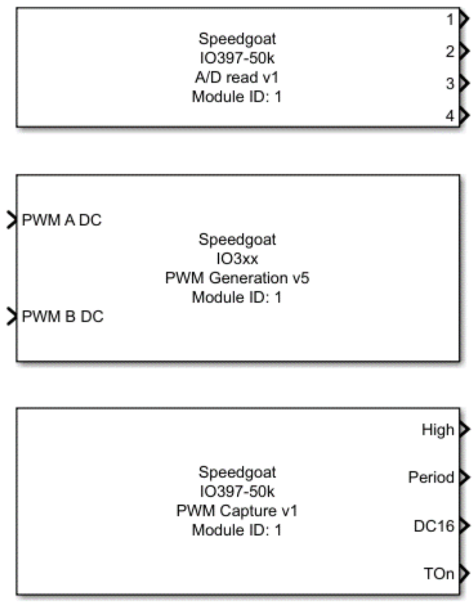


Figure 41. Used Blocks.

These blocks, such as Analog Input, PWM Capture, and PWM Generation, can be connected to standard Simulink blocks, including scopes, to facilitate comprehensive data management.

During the simulation, you have the option to set a specific simulation time if you want the simulation to run for a limited duration. This can be useful for testing specific scenarios or analysing the system's behaviour within a defined timeframe. However, if you don't set a maximum simulation time, the simulation will continue indefinitely until you manually stop it.

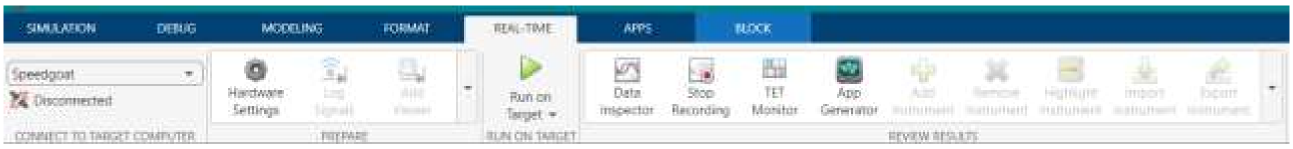


Figure 42. Simulink Real-Time Options.

3. Components Installation

In this chapter, we will focus on the installation of the selected components on the vehicle chassis with the objective of maximizing space utilization and ensuring easy disassembly of only the necessary parts. Our aim is to preserve the original setup for future projects while leaving non-interfering components in place.

The components to be added include the ePowertrain, consisting of the eMotor, planetary gearbox, and coupling plates for connection. Additionally, we will install the eMotor controller and the battery.

The installation process begins with the removal of the internal combustion engine (ICE) and the fuel tank from the vehicle chassis. Following the removal, CAD models are designed to create a digital representation of the vehicle. These models are essential in visualizing the vehicle and aid in the planning and installation of the ePowertrain components. In particular, the CAD models help in designing coupling plates that will facilitate the proper connection of the eMotor, planetary gearbox, and the existing driveline. Next, consideration is given to determining the optimal solution for fixing the gears on the planetary gearbox. The goal is to ensure efficient power transmission to the existing driveline. Lastly, the wiring from the receiver to the eMotor controller is studied.

By following this sequence of steps - ICE and tank removal, CAD modeling for visualization and design, studying the best solution for fixing the gears on the planetary gearbox, and analyzing the wirings from the receiver to the controller - the components can be installed on the vehicle chassis optimally. This process prioritizes space utilization, efficient power transmission, and reliable communication for the successful integration of the ePowertrain components.

3.1. ICE and Tank Removal

Starting from the baseline configuration of the Losi 5ive-2.0, the tank and the ICE (highlighted in yellow) are removed, they are not needed anymore for the final project and the occupied room is thought to be occupied by the ePowertrain components.

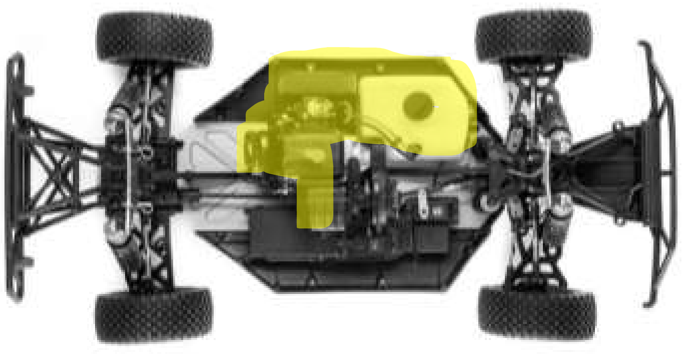


Figure 43. Losi 5ive-2.0 no body.

In a second time, the leverage system used for the ICE throttling valve control, is removed. This system includes a servo motor connected to it. However, since the servo motor is in a non-interfering position and its removal is not necessary for the installation of the ePowertrain components, it is left in place.

It is possible to see in the following picture the described system, there are three black arms with anodysed blu terminals, two are connected to the braking system exploited by the vehicle, the longest one connected to the ICE. It is worth nothing that the eMotor will also accomplish the brake task, to bettere underline the possibility to remove this system.

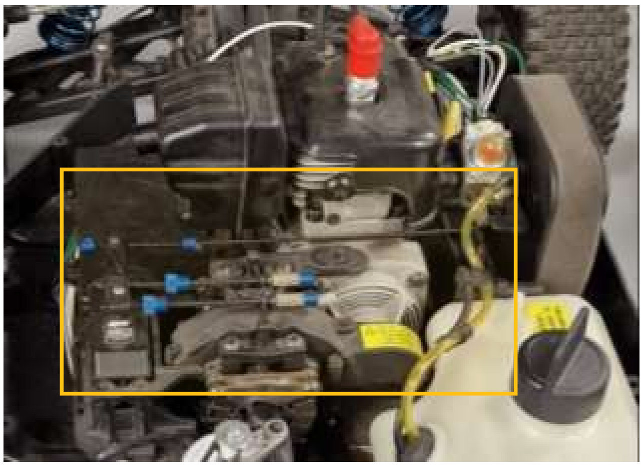


Figure 44. Throttle/brake control leverage.

Below the arms, there is a central differential plastic cover that is removed to facilitate the proper connection to the ePowertrain components. Some pictures are available to illustrate this process. As a result of the component removal, a significant amount of empty space is now available for the remaining components. The pictures demonstrate the outcome, showcasing the spacious area that can accommodate the installation of the ePowertrain components.

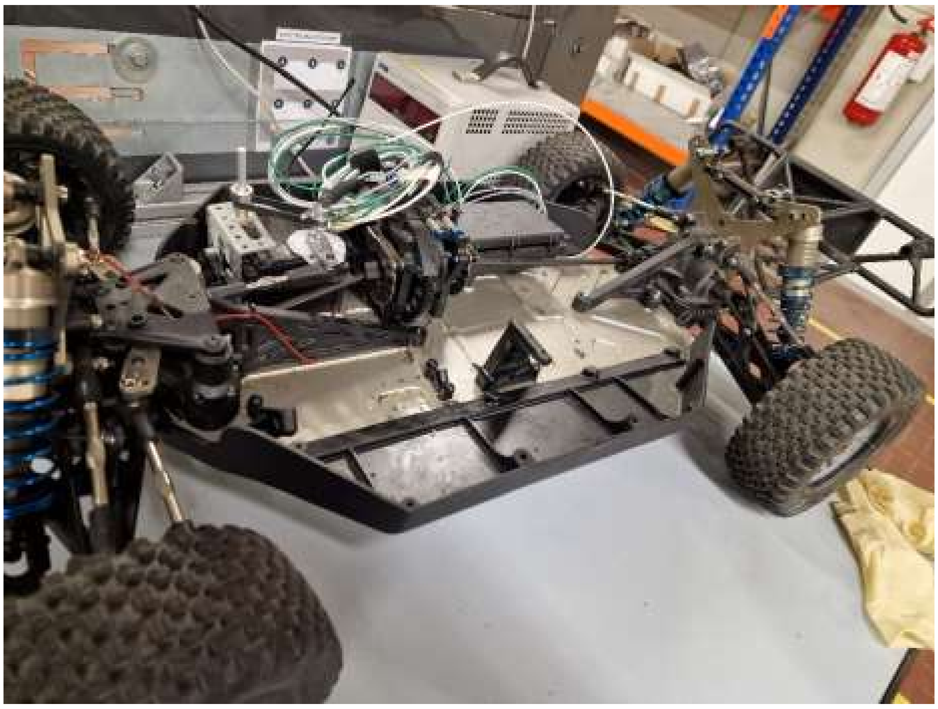


Figure 45. Chassis without ICE and Tank.

3.2. Power Transmission

The original ICE transmits power from the clutch to the central differential, being it four-wheel-drive, thorugh a 20 theeth gear, as shown in the Figure 48. ICE 20T gear.

Aiming to have the right connection among the ePowetrain and the driveline the gear is dismounted from it and a way to securet on the planetary is studied.

As a first attempt it is tried to avoid any machining on the toothed wheel, connecting it by a screw fixed on the top of the reducer, adding some threadlocker to better reduce the unscrewing effect coming from the reverse torque acting during the braking phase, as shown in Figure 46.

Since it was not enogugh, to prevent this to happen, it is decided to employ a key, taking into account that a key shaft slot was present on the gearbox. To do that a slot on the wheel is drilled by a workshop. It is possible to see that in Figure 47. For sake of safety a screw is still present, as in the first attempt, but now it is not anymore in charge of torque transmission, just to hold in place the gear alongside the shaft.

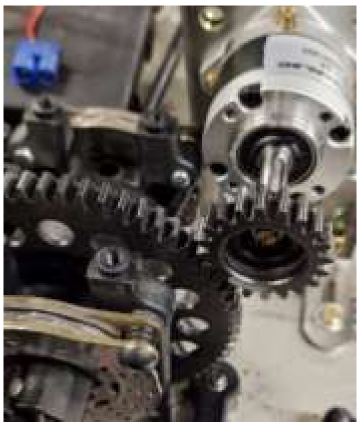


Figure 46. Gear Planetary Screwed.

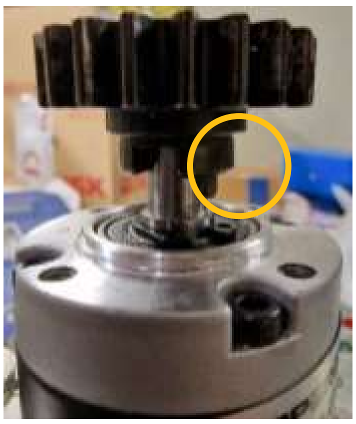


Figure 47. Gear Key Connected.

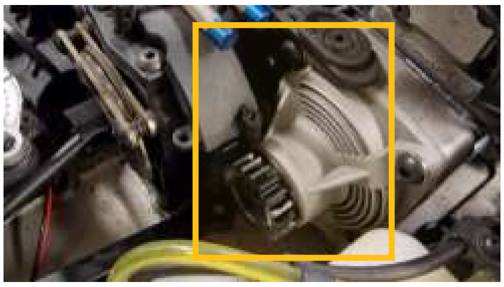


Figure 48. ICE 20T gear.

3.3. ePowertrain Assembly

The eMotor works in a 0-4000rpm speed range, with a peak torque of 0.7 Nm; to improve this value and better manage the torque-speed characteristic the selected planetary gearbox is added.

The planetary gearbox follows a NEMA 17 standard, what determines the screw mounting holes position; on the other hand, the eMotor does not, so a way to couple them together is studied, the pictures summarise what said.

The solution is to employ two off the shelf coupling plates, Figure 51, drilling the needed holes. These components are 3mm thick drilled iron alloy plates. Being the already existing holes not compatible with the powertrain components request, it is decided to drill some of them in the right place, exploiting the central area, in which no holes are present.

Since a certain precision is needed, to perform a good job and reduce the error margine, some CAD models were developed well examinated in the CAD Models section.

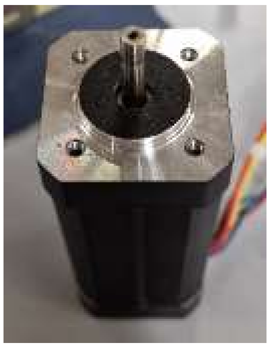


Figure 49. Planetary Gearbox mounting holes.

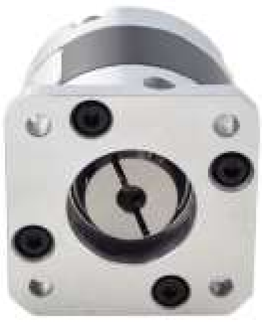


Figure 50. eMotor mounting

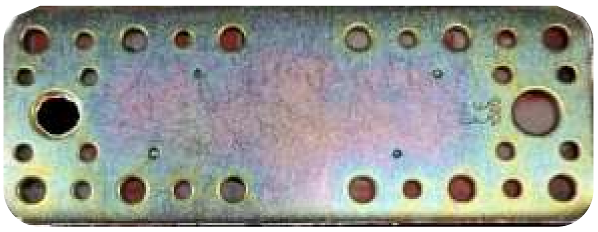


Figure 51. Coupling Plate Off the Shelf.

The assembly process continues by attaching one plate to the planetary gearbox using four screws, as depicted in the accompanying picture. Similarly, the other plate is affixed to the eMotor following the same method as previously described.

Once the plates are secured to their respective components, they are tightened together through the existing lateral holes. These screws not only hold the system in place but also connect the L-shaped mounting plates, which are responsible for securing the entire assembly to the chassis.

Due to the eMotor shaft being longer than the input hole of the planetary gearbox, there is a remaining clearance between the plates. To prevent excessive bending and ensure stability, nuts are employed as spacers. These nuts help reduce the gap between the plates, providing additional support.

To optimize space utilization and save time, the previously mentioned additional plates undergo a modeling process, followed by careful selection and purchase for integration into the assembly. These plates serve specific purposes and contribute to the overall functionality of the powertrain.

In the Figure 53 is possible to have a look on the complete powertrain.

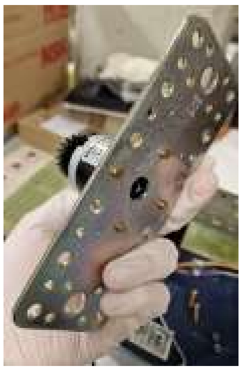


Figure 52. Planetary Gearbox Coupling Plate.

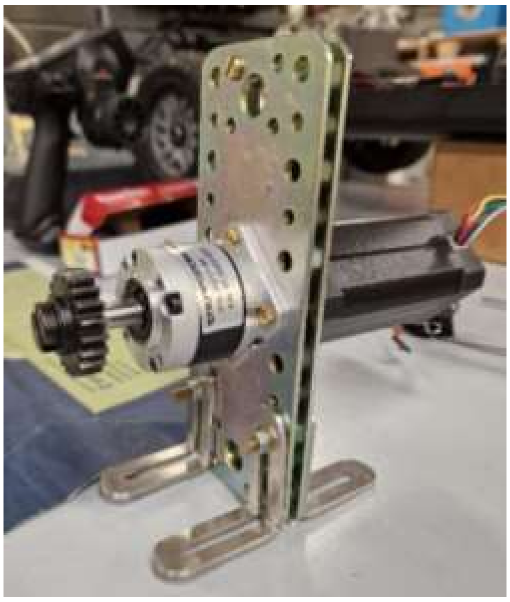


Figure 53. Final Powertrain.

3.4. Chassis connection

Once the powertrain assembly is completed, the next step involves connecting it to the chassis using L-shaped coupling plates.

In selecting the appropriate 90° bent plates, there are multiple options to choose from. The use of CAD (Computer-Aided Design) models proves to be highly advantageous during the selection process. The decision to purchase is solely based on the dimensions provided in the technical sheets. After careful consideration, the plates shown in Figure 54 are determined to be the most suitable due to their flexibility, potential for future modifications, and efficient space utilization. These mounting plates stand out because, instead of having a series of holes, they feature a slot along the main dimension. This slot proves to be highly advantageous in accommodating layout variations.

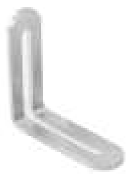


Figure 54. L-Shaped Coupling Plate.

In the initial stages, an attempt was made to connect the entire powertrain to the chassis using the existing holes that were originally designed for engine mounting. Figure 55 illustrates the presence of four slots already in place. However, this approach was ultimately rejected. The inner part of the slots was larger than the base of the ePowertrain, resulting in a misalignment of the system in relation to the chassis. Consequently, the powertrain ended up being slightly rotated along the axis passing through the shafts. This lack of perpendicularity could potentially compromise the stability and solidity of the coupling.

To address this issue, a decision was made to expand one of the two slots being used, as depicted in Figure 56. This adjustment ensured a proper positioning of the ePowertrain, rectifying the misalignment and allowing for a secure coupling with the chassis.

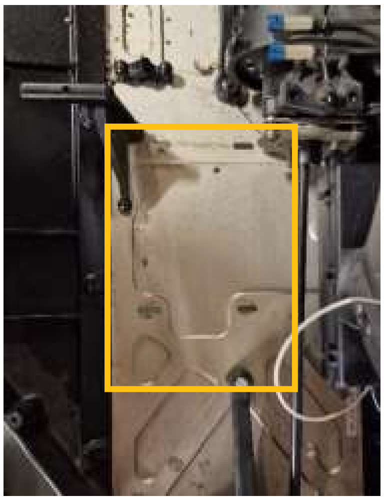


Figure 55. Chassis Engine Mounting Slots.

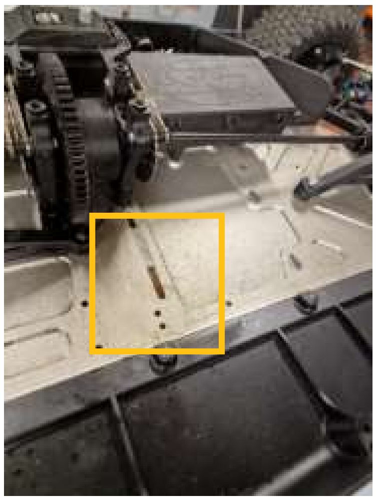


Figure 56. Expanded Slot.

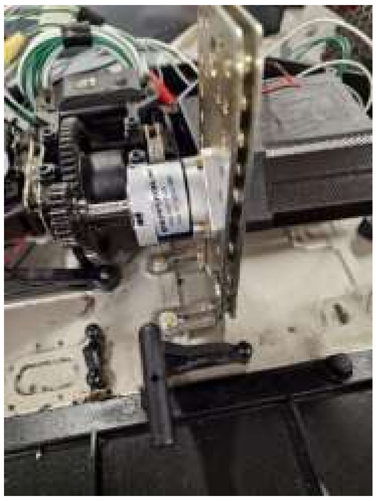


Figure 57. First attempt Powertrain Coupling.

The current approach involves using only two L-mounts on the front side for the initial coupling, resulting in a somewhat unstable system that is susceptible to fluctuations from the driveline. It is worth noting that, at this stage, the gear is not precisely centered and is only tightened with a screw, to overcome this the transition to key fixing gains relevance aiming to improve stability.

To achieve a more stable system, two additional L-shaped plates are being added at the back without drilling holes in the chassis. Instead, they are placed on a damping material. Furthermore, a vertically adjustable support is being introduced to ensure system stabilization. This support can be adjusted in height using a screw system. The same damping material used beneath the mounts is being wrapped around the eMotor, providing support for the stabilizer.

By incorporating these additional L-shaped plates, damping material, and a vertically adjustable support, the aim is to achieve a more stable system that minimizes vibrations and enhances overall performance.

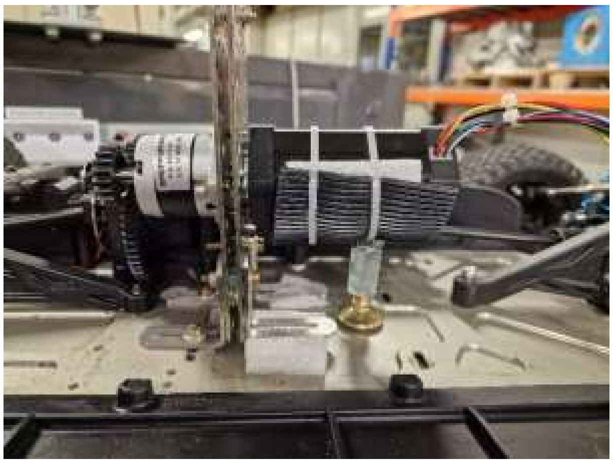


Figure 58. Final ePowertain Coupling.

3.5. CAD Models

During the project, CAD models are created using the SolidWorks software[13]. These models proved to be highly beneficial, serving two main purposes. Firstly, they facilitated the future development of a complete digital vehicle model. Secondly, they provided a clearer understanding of the space occupied by various components, offering an improved overview of the overall layout.

Aiming to study a way to connect the planetary to the eMotor, this last was the first component designed. The greatest part of the dimensions come from the technical sheets, while the missing ones were taken by hand thanks to a caliper or a meter. Following the planetary gearbox and then the coupling plates were modelled.

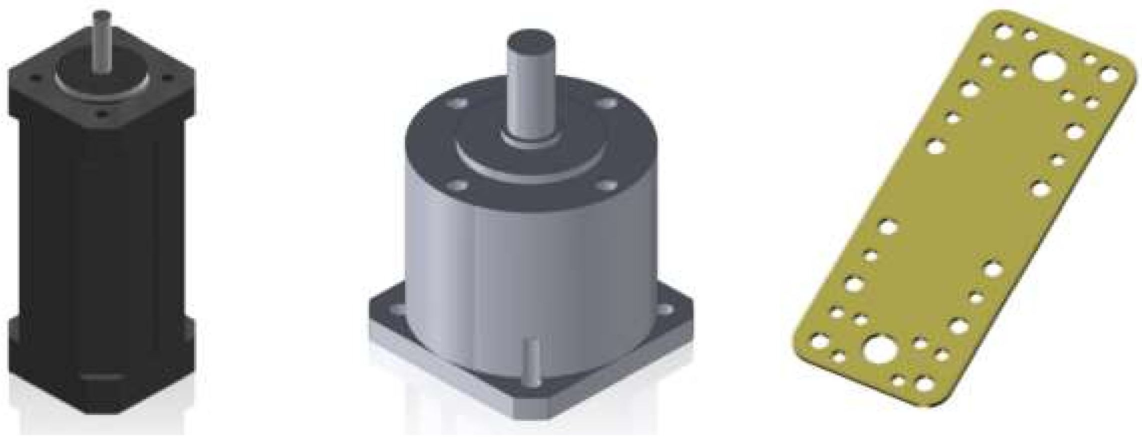


Figure 59. eMotor, Planetary Gearbox and Coupling Plate CAD.

The next step is to add the necessary holes to the two coupling plate models. Both plates need a central common hole for the eMotor shaft, surrounded by four smaller ones specific of the device being linked. Once obtained the CAD, the mechanical drawings are obtained and given to the technician for the processing. The results are show in the following pictures.

[13] By Dassault Systèmes.

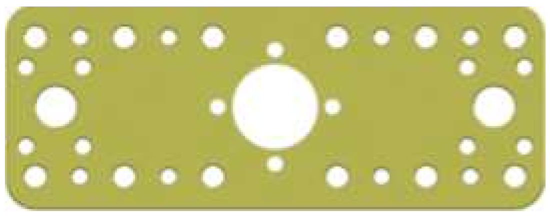


Figure 60. eMotor Coupling Plate CAD.

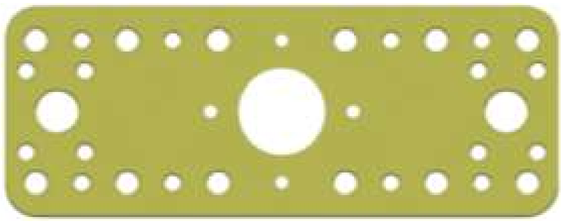


Figure 61. Planetary Gearbox Coupling Plate CAD.

Once all the components were obtained, an assembly of the ePowertrain was carried out as shown in Figure 62. This assembly only requires two M4 screws, connected to their respective nuts, while the other two screws are added during the connection with the chassis later.

An existing chassis model was modified to incorporate the missing ICE mounting holes and two cylinders to represent a specific occupied volume, primarily from the driveline supports. The positions of these elements were determined manually due to the lack of a technical drawing.

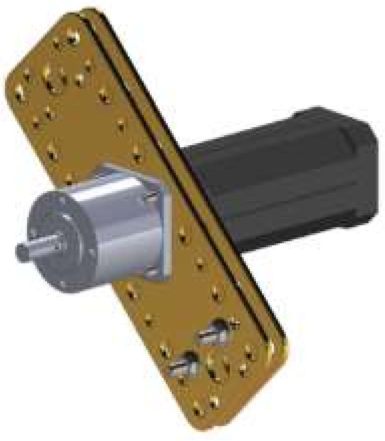


Figure 62. eMotor Assembly CAD.

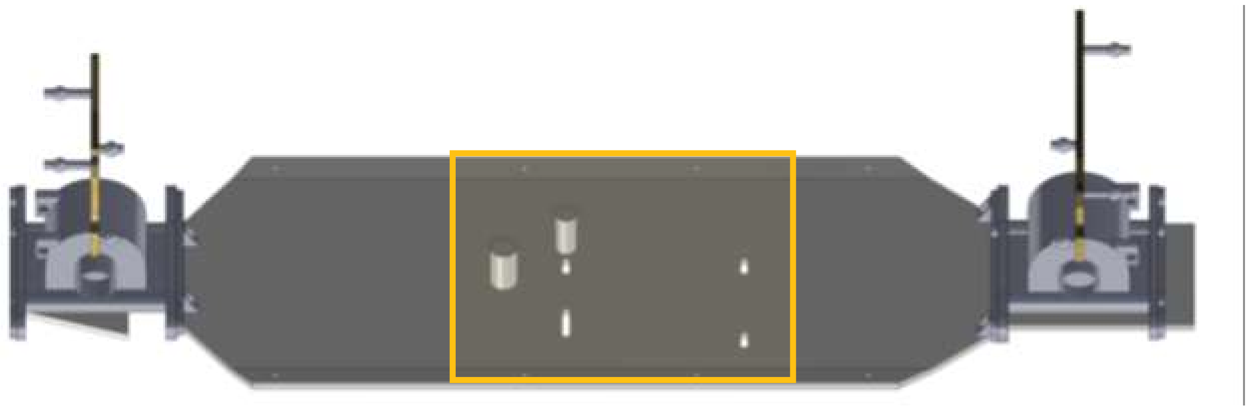


Figure 63. Chassis CAD Highlighted Additions.

The chassis model proves to be highly beneficial when it comes to selecting the L-shaped mounts. These mounts are available in online shops, providing a wide variety of options. The actual purchase of the mounts is made at the end of the selection process. To choose the compatible mounts, the assumption is made that no mechanical processing is required. The CAD creations enable a better estimation of the occupied volume, as it becomes challenging to perceive the actual occupied space solely by focusing on the dimensions.

Below are the different mount options grouped together, and the chosen option is the one positioned at the top-left, as mentioned in the previous section.

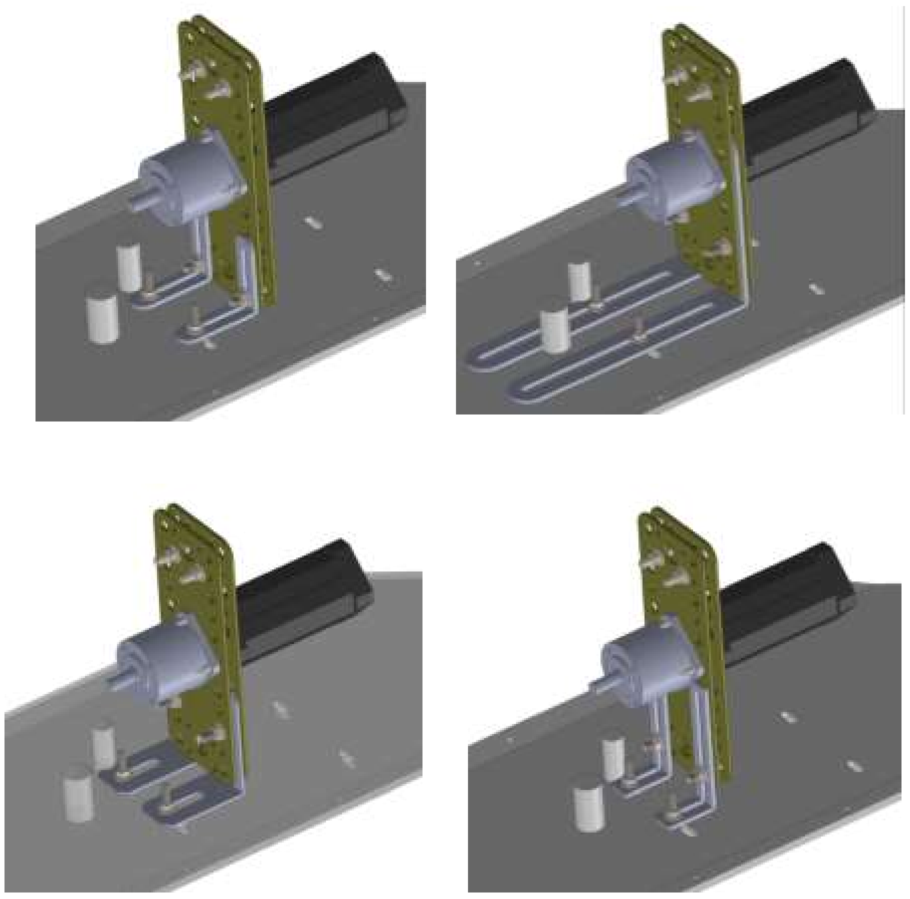


Table 4. L-Shaped plates options.

The last picture represents a model that includes the mounts on the back side of the ePowertrain and the employed battery.

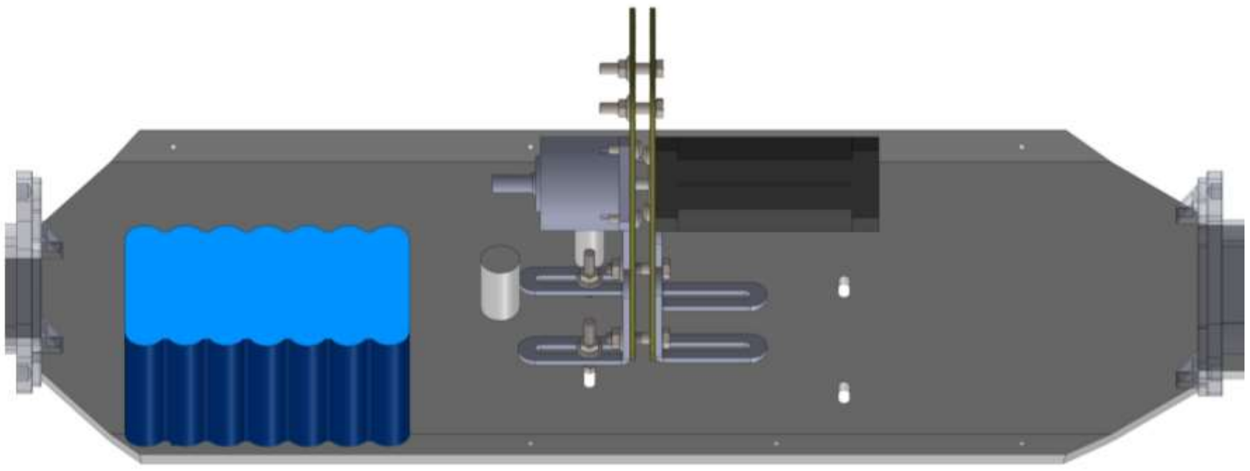


Figure 64. Complete CAD Model.

4. Experimental Tests

In this chapter, a detailed explanation of the tests performed for the tuning and optimization of the controller will be provided. It will cover the system layout, devices used, settings, and speed signals employed during the tests. The work conducted will be grouped into specific cases, which will effectively collect and present the tests performed.

In each test, the control flow follows a similar pattern. The eMotor is controlled by the controller, which regulates the timing of the coil power to achieve speed regulation. The main focus of the tests is to fine-tune the parameters related to speed control, although it should be noted that the controller is also capable of performing torque control.

This motor includes a Hall sensor what is in charge of recognising the rotor position to exploit a better closed loop control; it is composed by three sensors, one for each phase, producing a square wave when the rotor passes through one of those. This signal is managed by the controller, what also provided the 5V needed by the sensor.

The controller needs a reference signal what is provided in different ways, exploiting the Arduino Portenta H7 board, a potentiometer, the vehicle receiver and lasty implying the Real-Time Target Machine Speedgoat. The signals were both digital (PWM) than analogical.

The core principle of eMotor control remains consistent, while the variations lie in the pathway, feedback information, devices used (such as switches, Arduino, potentiometer), and the method of energy supply. Of course, the best solution to control the vehicle is the wireless one, the other requires physical connection from the device in charge of management to the controller, loosing effectiveness for the purpose of using the vehicle to study the dynamics. The most complex and precise solution is, instead, associated with the Speedgoat system. Thanks to its advanced features and available options, the Speedgoat system provides extensive capabilities for controlling and studying the dynamics of the entire vehicle system. It offers a comprehensive and versatile platform for conducting detailed and sophisticated control experiments and analysis.

During the initial stages of testing, the powertrain underwent a series of bench tests. The motor itself was tested first, followed by testing the complete powertrain. Once the powertrain was securely installed on the vehicle chassis, the vehicle was lifted off the ground to ensure that the wheels were not in contact with the surface. This allowed the wheels to rotate freely. Finally, when the vehicle was capable of movement, it was released to move around the laboratory without any constraints.

Being the battery purchase process longer than expected the power during the bench tests was supplied by a static device. Once the battery arrived the whole powertrain was tested with this new electric source and then, checked everything fine, added to the vehicle, becoming the only way for energy providing, even when the vehicle was bench tested.

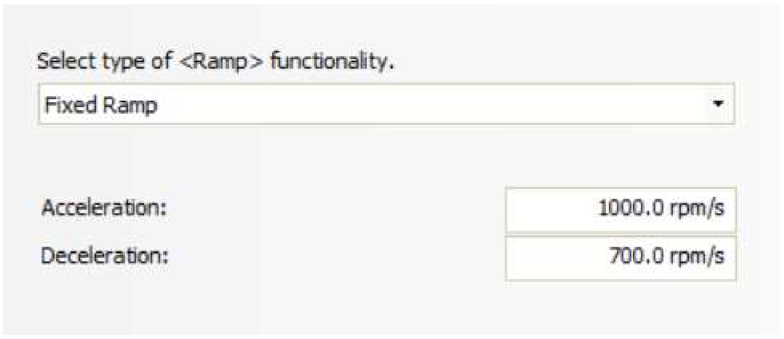
For safety reasons, the controller necessitates a digital signal to activate the eMotor. This signal serves to prevent any unintended movements, particularly during driveline works or maintenance activities. Additionally, this digital signal can also be utilized to determine the rotation direction of the eMotor, as demonstrated in Case 2 by means of a switch.

It is worth noting that certain eMotors may exhibit scattered or erratic movements at speeds below a specific threshold. In the case of this eMotor, after a trial-and-error process, it was determined a 61 rpm. To address this, the controller settings can be adjusted accordingly, which is the best approach. Alternatively, as done in Case 1, where the enabling signal comes from the Arduino board, can be switched off by incorporating an "if" statement into the sketch, to switch the enabling signal to the LOW voltage level, at which the controller inhibit the eMotor movement.

In all cases, the eMotor parameters remained consistent. This includes parameters provided by the datasheet, such as maximum speed, current, and other specifications, as well as parameters determined through the autotuning procedure.

An asymmetric acceleration/deceleration ramp for the eMotor was configured based on Figure 65. Controller Set Ramp. However, this ramp was no longer utilized in the tests conducted for the 2nd case, as the investigation focused on identifying the cause of the discrepancy between the desired and actual eMotor speed.

The ramp feature serves as an effective method for managing the speed transitions of eMotors, particularly in large systems with significant inertias. It helps avoid sudden and undesired reactions, thereby enhancing the overall system behaviour and reducing stress and wear on components. Although it was not required for the specific tests in the 2nd case, it was included to assess its impact on the system under test conditions.


Select type of <Ramp> functionality.
Fixed Ramp

Acceleration: 1000.0 rpm/s
Deceleration: 700.0 rpm/s

Figure 65. Controller Set Ramp.

4.1. Case 1 – Static Test

This initial case, conducted during the early stages, focused on bench testing the motor/controller system. The objective was to gain a preliminary understanding of eMotor control by exploring various solutions, experimenting with controller parameters, and delving into the comprehensive implementation of the Escon Studio software.

For the tests, the key components utilized were the eMotor, its controller, a static power supplier, an Arduino Portenta H7 board, and a potentiometer. Since the battery had not yet been purchased and was not essential for the static tests, the static power supplier served as a substitute. However, the Arduino board played a crucial role in this setup. In this configuration, the PC was an integral part of the loop, as it was not an offline working layout.

Basically, it was a step-by-step complexity increase in discovering and exploiting the controller features effectively. The speed was controlled by a PWM signal produced by Arduino, firstly thanks to a fixed value, written during the code development and than exploiting a potentiometer to vary it sequentially. Additionally, the controller's analog speed signal feedback was tested.

4.1.1 Controller/Arduino Configuration

After encountering some incorrect configurations, the controller was successfully set up by following the procedure outlined in the the The configuration and interfacing of the controller with the eMotor is performed using the Escon Studio Software, developed by Maxon. The controller is connected to the software via the micro-USB port, allowing for convenient setup and configuration.

Getting Started section. The specific steps involved in configuring the eMotor parameters have already been discussed and will not be reiterated in this context.

The configuration began selecting the speed control working mode, the enabling signal came by the Arduino board, it is produced at the D6 pin and sent to the 2nd position of the Digital Plug controller connector, it just enables the rotation according to the polarity, since Arduino is able only to fed positive voltage it enables the counterclockwise direction, as shown in the figure.

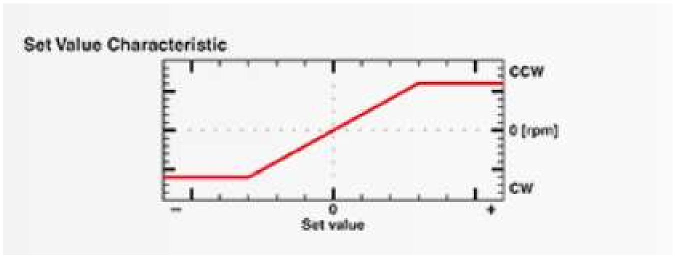


Figure 66. Enabling Signal Characteristic.

In the subsequent stage, the speed reference signal is tuned, utilizing a PWM signal generated by the Arduino Portenta H7 board. This further emphasizes the crucial role of this PCB during the initial phases of the setup. The controller is capable of reading PWM duty cycles within the range of 10% to 90%, and as a result, the speed thresholds for both the controller and the Arduino board are adjusted accordingly.

In the case of the Arduino PWM signal, duty cycle values are represented within the 0-255 range, as it utilizes 8 bits. In order to establish a properly functioning system, the 0-4000rpm range for the eMotor must correspond to the 10-90% duty cycle values. Converting these percentages into the appropriate PWM values involves rounding up or down to the nearest whole number.

For the lower threshold, the 10% duty cycle corresponds to approximately 24.6 bits. Since decimal values cannot be selected, the nearest higher integer is chosen, which is 25. This value corresponds to a duty cycle of approximately 10.2%.

Similarly, for the upper threshold, the 90% duty cycle corresponds to approximately 230.4 bits. Rounding down to the nearest lower integer, the value selected is 230. This corresponds to a duty cycle of approximately 89.8%.

By establishing these specific duty cycle thresholds based on the desired speed range, a proper correspondence between the eMotor speed and the PWM duty cycle values is achieved, ensuring accurate control and operation of the system.

“`pwm=map(speed, 0, 4000, 25, 230); //PWM range merged with controller range`”[14]

[14] eMotor Controller Enabling Signal with Controller Feedback, code line 27.

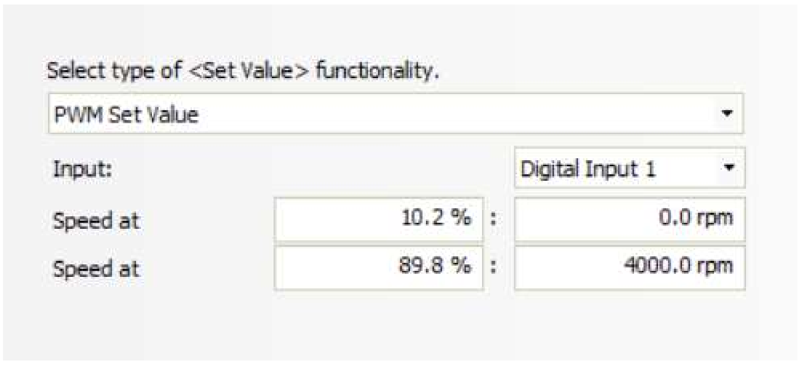

Select type of <Set Value> functionality.
PWM Set Value
Input: Digital Input 1
Speed at 10.2 % : 0.0 rpm
Speed at 89.8 % : 4000.0 rpm

Figure 67. PWM Controller Thresholds.

As already said the speed ramp is there used. No minimal eMotor speed was set because this was taken into account by the Arduino codes. The first two codes of the eMotor Sketches chapter, exploits a fixed speed selected during the writing phase, so no matter about the minim speed, it is just reminded to stay over the 61-rpm limit.

On the other hand, in the 3rd code, a potentiometer, fed by the Arduino 3V3 (Vcc) pin, is used to tune vary the speed. The potentiometer is connected to the Arduino board sending an analogical value, variable according to user request, what is translated to a PWM duty cycle and sent to the controller.

Lastly, the controller is set to send-back an analogical value, by the 3rd pin of the Controller Analog Plug, representing the motor actual speed (filtered. The analogical value could be adjusted in a 0 to 4V range as desired, since the board is able to read analogical signal up to 3V the range is tuned as follows:

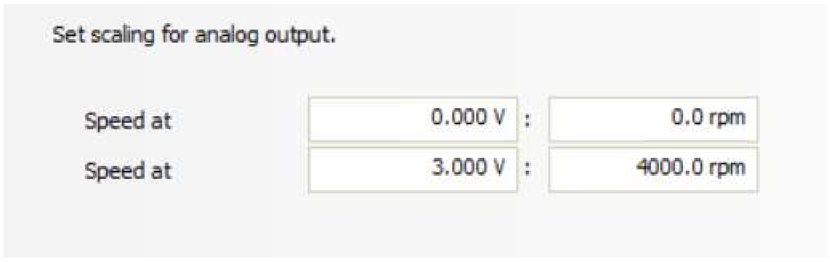

Set scaling for analog output.
Speed at 0.000 V : 0.0 rpm
Speed at 3.000 V : 4000.0 rpm

Figure 68. Controller Analogical Speed Feedback.

This signal is read by the Arduino, converted thanks to the map command, and printed to the Serial Monitor.

```
analogReadResolution(16); //now the range should be 0-65535
readvalue=analogRead(analogspeedpin);
clspeed=map(readvalue, 0, 65535, 0, 4000);

Serial.println(clspeed); [15]
```

This layout was not thought to be used offline, mainly due to Arduino, fed by the PC to which is connected, is possible to overcome this condition. The board could be powered in different ways, as explained in the Power the Arduino chapter, a method is to provide a voltage level to the Vin pin. The controller could provide, through the Analog I/Os plug, a fixed tension level, as requested during the setup phase, to feed the Arduino board aiming the system to work without the need of a PC. It is worth noting that the controller is fed thanks to the power supply system.

[15] External Potentiometer Regulation and Controller Feedback, code lines 30-34.

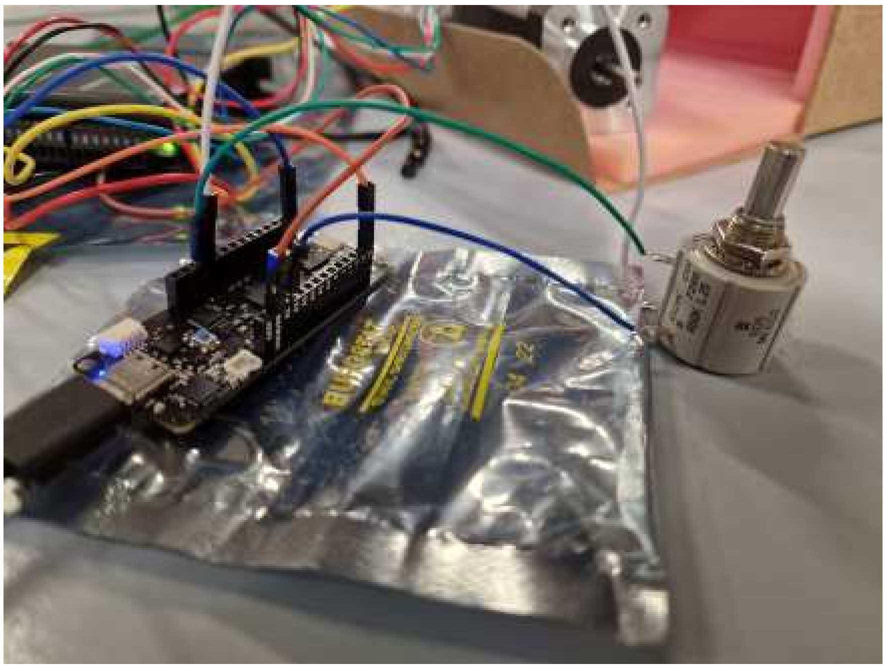


Figure 69. Potentiometer-Arduino Link Focus.

4.2. Case 2 – Potentiometer Control

In this test case, a transition from a discrete digital PWM signal to a continuous analog signal was made, and it was grouped and discussed as part of the control analysis. The primary motivation for this stage was to identify the source of discontinuity that caused a mismatch between the speed signals obtained from the potentiometer (connected to the Arduino) and the feedback voltage level provided by the controller. Addressing this issue was important for accurately comparing the voltages or speeds, which are essentially mapped voltage levels. Additionally, this stage aimed to explore other controller features to enhance powertrain control and, consequently, the vehicle.

The devices involved in this particular case were different. The role of the Arduino changed from being the source of the PWM signal to serving as a reading device. It was used to read and compare the analog values from both the potentiometer and the controller. The potentiometer remained in the setup, connected to both the Arduino and the controller. Additionally, a switch was utilized.

To record the values, a PC was integrated into the loop. However, it's important to note that controlling the eMotor via the potentiometer does not require the PC and can function offline, transitioning from a static power supply system to a battery-powered setup. The data were recorded using the CoolTerm software [1], emulating the Arduino Serial Monitor.

To facilitate real-time visualization and comparison of the potentiometer and controller values, a bridge wiring configuration was designed. Figure 70 illustrates this setup, although it may not be entirely clear due to the large number of connections. A more schematic depiction can be found in the wiring section of Case 2 Additionally, in the picture, a multimeter is visible, used to investigate any voltage drop between the compared signals, as it was initially suspected to be responsible for the discrepancy. However, it was determined that the voltage drop was not the cause of the mismatch.

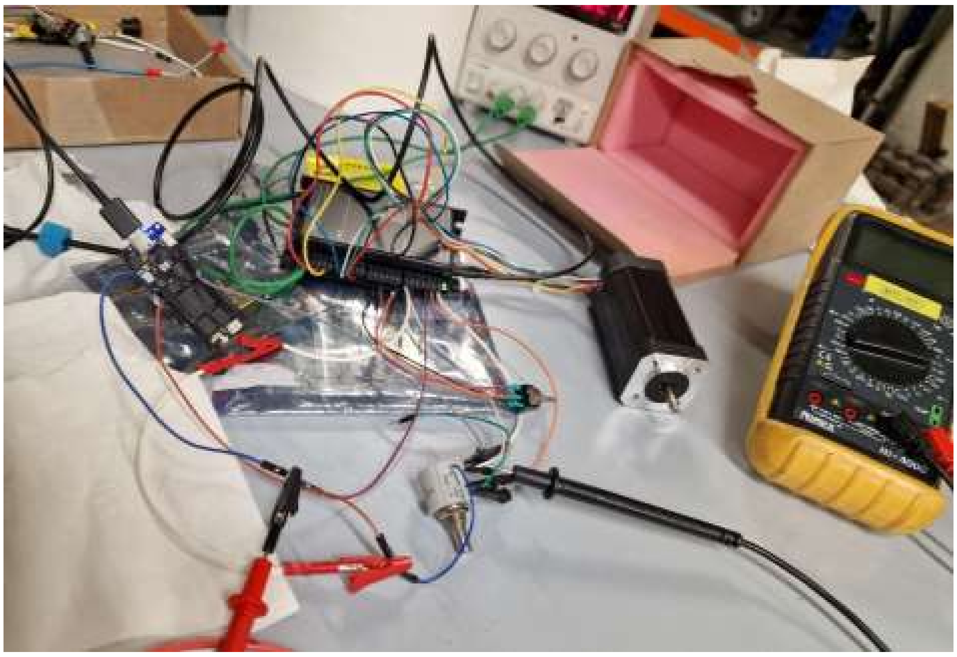


Figure 70. Case 3 Bridge Connection, Multimeter in the Loop.

Since a lot of connections need to be done, it was truly helpful to use the crocodile connector wires, shown in Figure 71. Even if not shown, the terminals are on both wire sides allowing a smart and flexible connection, as in the upper situation in which a potentiometer was exploited.

The enabling signal didn't come from the Arduino board anymore, the switch was charged of doing so.

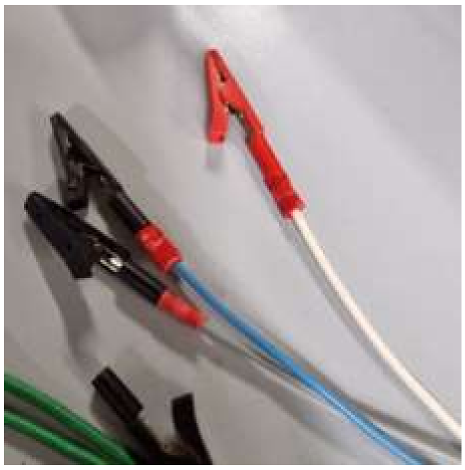


Figure 71. Crocodile Connector Wirings.

Overall, this test case involved transitioning from a digital PWM signal to an analog signal and focused on identifying the source of discontinuity in the speed signals. The devices used included the Arduino in its reading role, the potentiometer, and a switch. Data recording was facilitated by the PC, and a bridge wiring configuration was established to enable real-time visualization and comparison of the potentiometer and controller values.

4.2.1 Signals Unmatching

During the data plotting process, which was performed using the Arduino Serial Monitor or Serial Plotter, it was discovered that the speed signals exhibited incomplete overlapping. In the previous case, the control signal from the potentiometer underwent internal conversion by the Arduino board, utilizing the built-in ADC (Analog-to-Digital Converter). This conversion process reduced the reading resolution from 16 bits to the 8 bits utilized by the PWM signal. The converted signal was then sent to the controller for processing.

In an attempt to address the issue, the initial idea was to bypass the potentiometer signal conversion performed by the Arduino. Instead, the potentiometer was directly connected to the controller's Analog Plug J6, enabling it to regulate the speed independently. To ensure that the signal was also directed to the Arduino, a bridge connection was established. This involved linking the GND (Ground) pins of the controller, the potentiometer, and the Arduino together. Additionally, the regulated voltage from the potentiometer was directed to both the controller and the Arduino.

By implementing this bridge connection and allowing the potentiometer-regulated voltage to be received by both the controller and the Arduino, it was expected to improve the overlapping of the speed signals and address the discrepancy observed during the data plotting phase.

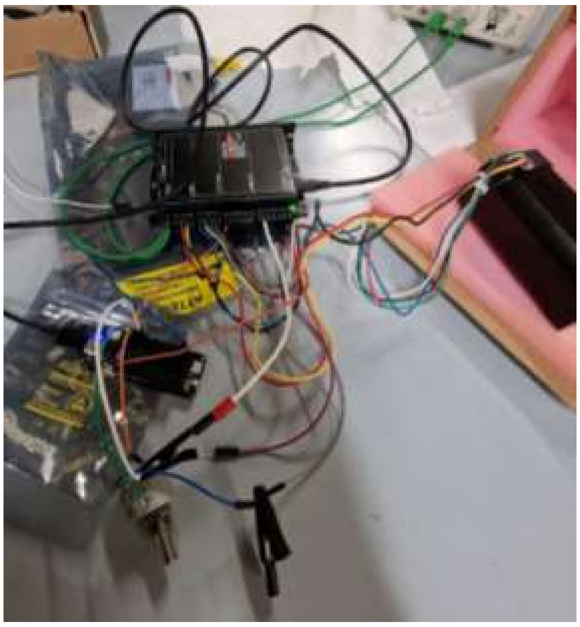


Figure 72. Bridge Connection Detail.

Even if the Serial Monitor is an effective and quick instrument, it is not effective to perform an in-depth review due to the few charts (data) visualisation options.

The software CoolTerm helped in overcoming this issue. Indeed, this software acts as an external Monitor to which the Portenta Board sends the desired data, in this case the speed signals; being able of doing so, it also provides a data recording feature, allowing to write those values into a .txt file. This software works only when the Arduino Serial is closed.

The recordings performed were different, to better understand the phenomenon, then plotted thanks to MATLAB. There was an issue related to the plots since they were full of spikes and didn't allow a fair comparison among signals, to smooth these a filter was created.

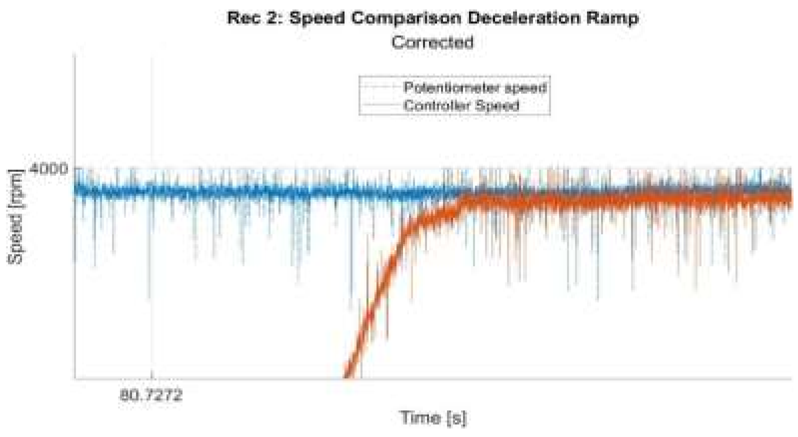


Figure 73. Spiked Data Plot.

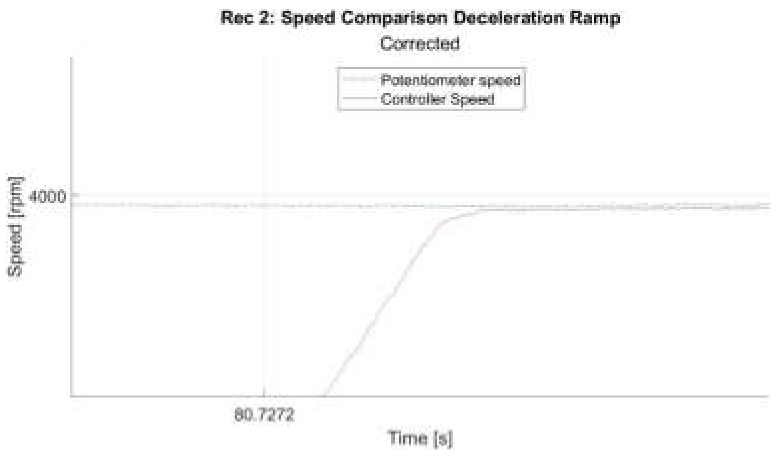


Figure 74. Filtered Data Plot.

For each setup two recordings were performed. The Test 1 recordings account for the Arduino Sketch correction, as discussed in the eMotor Sketches chapter, in attempt to address this inconsistency to a voltage loss of approximately 5 mV. This loss corresponds to around 100 bits, which translates in, more or less, 7 rpm, not enough to compensate for it. Additionally, this was a constant correction, not following the needs being its value speed dependant, this contributes to explain the wrong behind. In addition to the correction, the controller setup included the eMotor acceleration/deceleration ramp, set as previously shown.

In the plots is possible to appreciate the *potentiometer speed*, a pure mapping of the potentiometer voltage into the corresponding speed range. Similarly, the plotted *controller speed,* what is the analog voltage signal coming from the controller, mapped as previously done to the speed range. The potentiometer was manually adjusted to regulate the speed, allowing for a comprehensive study of the differences in behaviour while transitioning from maximum to minimum speeds.

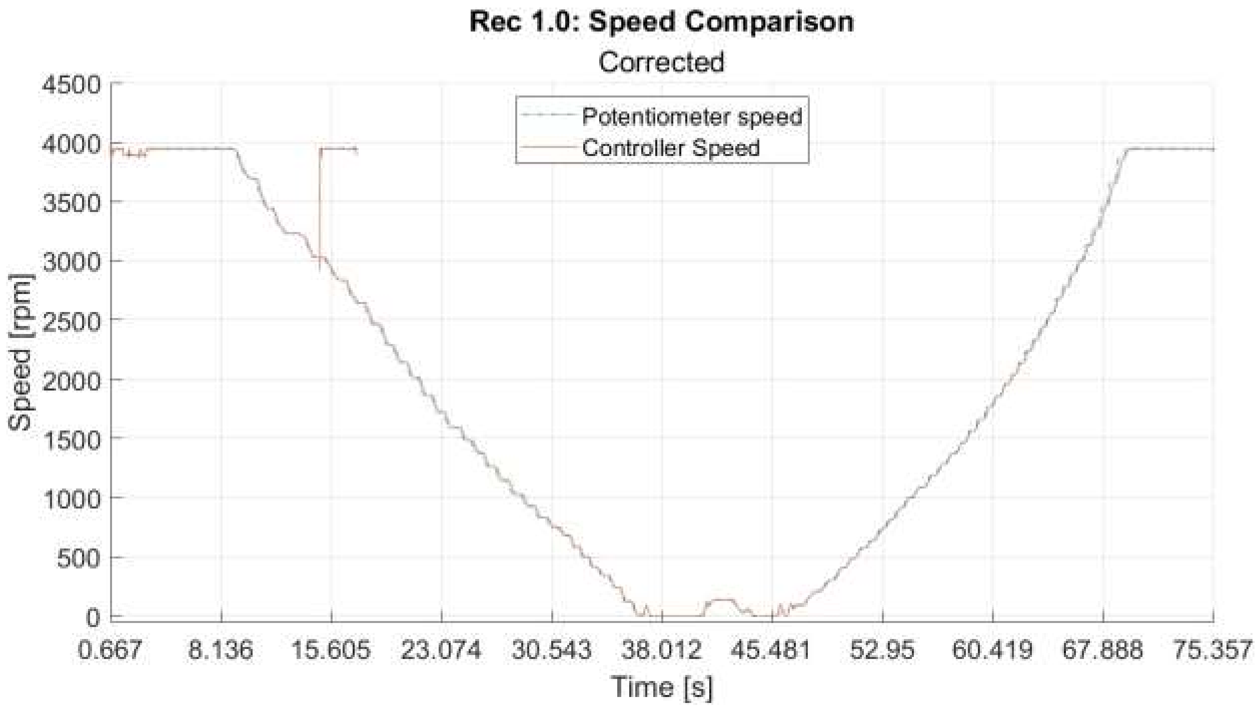


Figure 75. Test 1 Recording 1.

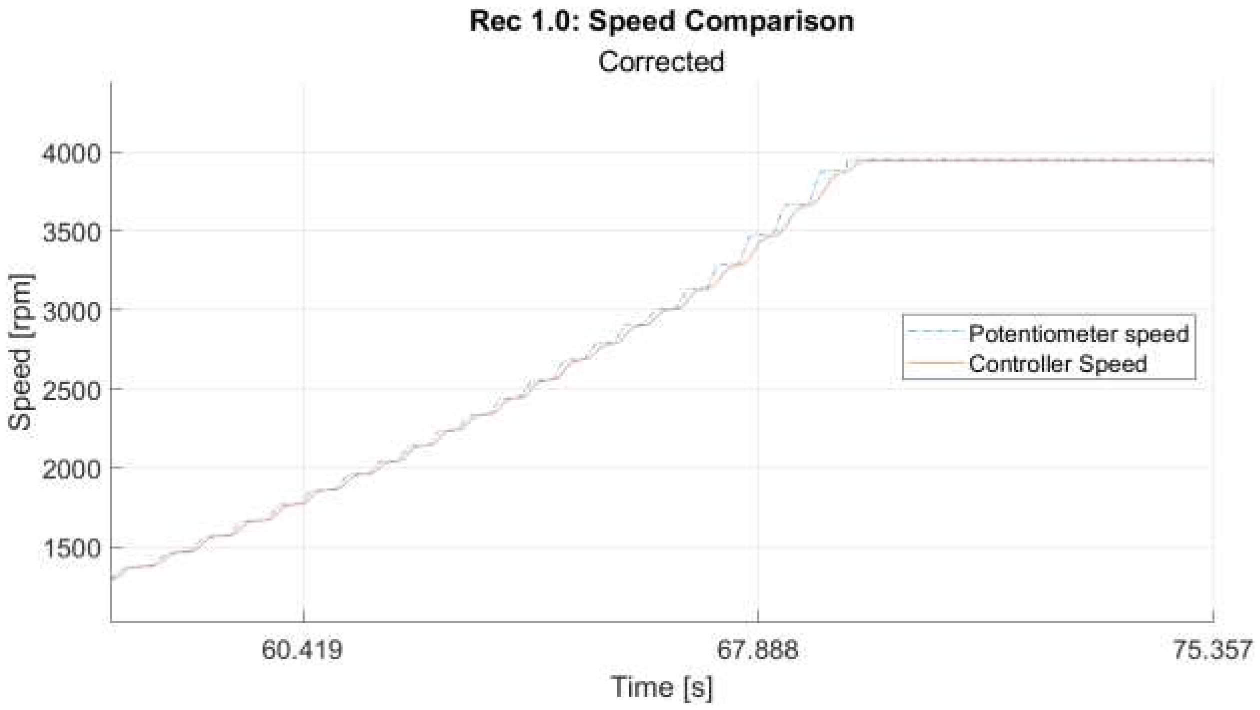


Figure 76. Test 1 Recording 1, Signals Detail.

The Test 1 second plots accounts for the same controller/sketch setup, in this case no potentiometer adjust was done, as the blu dashed line explains. Here the switch, charged of enabling signal production, was turned on and off starting from the maximum speed available. It is clearly visible the influence of this parameter especially during the acceleration phase.

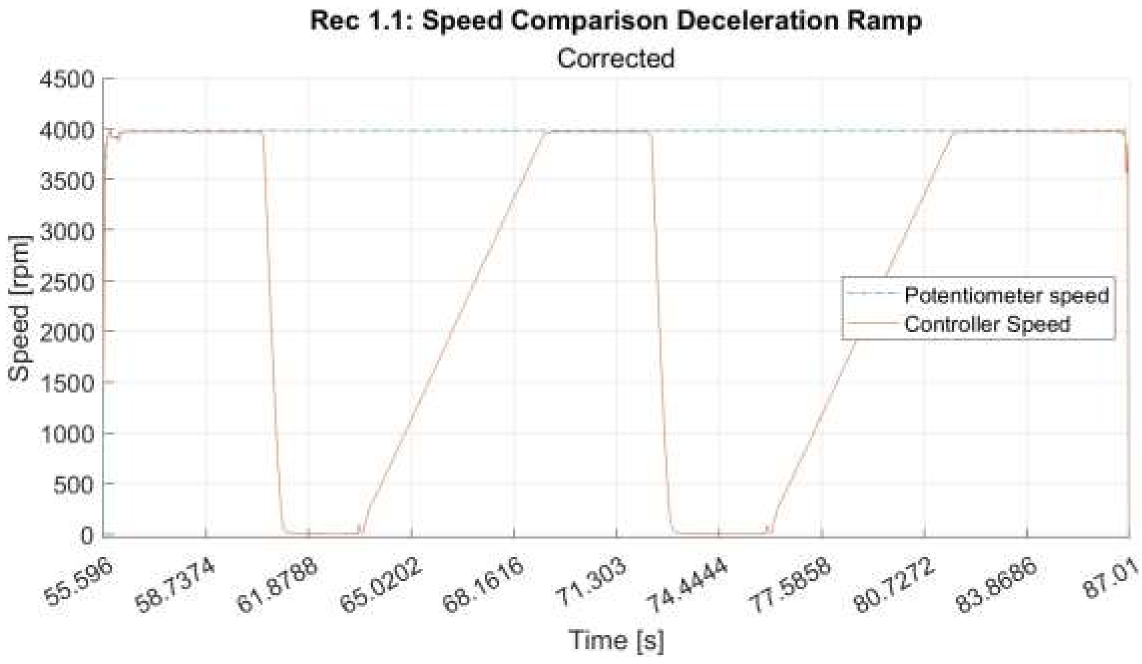


Figure 77. Test 1 Recording 2.

The Test 2 was performed without any manual correction added to the sketch, but the ramp was still considered. It is possible to appreciate a huge difference among signals along the whole plot. As in the previous test the speed variation came from the manual tuning of the potentiometer.

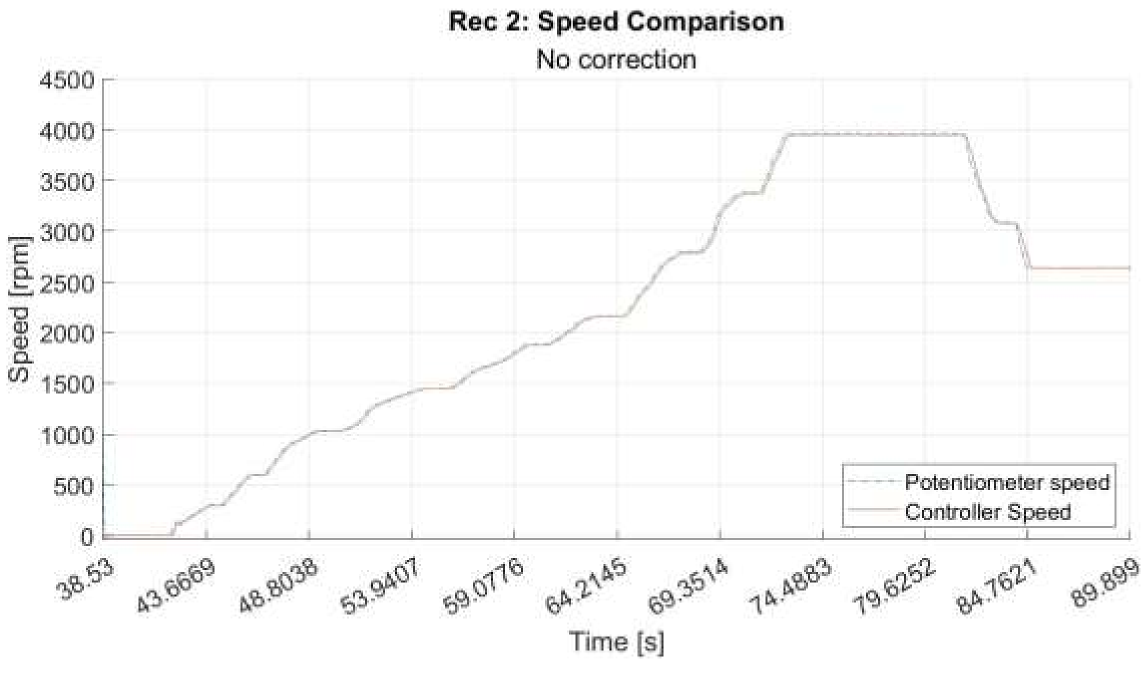


Figure 78. Test 2.

On the speed ascending side, the eMotor needs a certain time to match the potentiometer request; this behaviour is symmetrically in the descending part. This was both attributed to the controller ramp set while the eMotor inertia. It is evident that the discrepancy was almost disappeared at lower speed, as shown by the last right straight line.

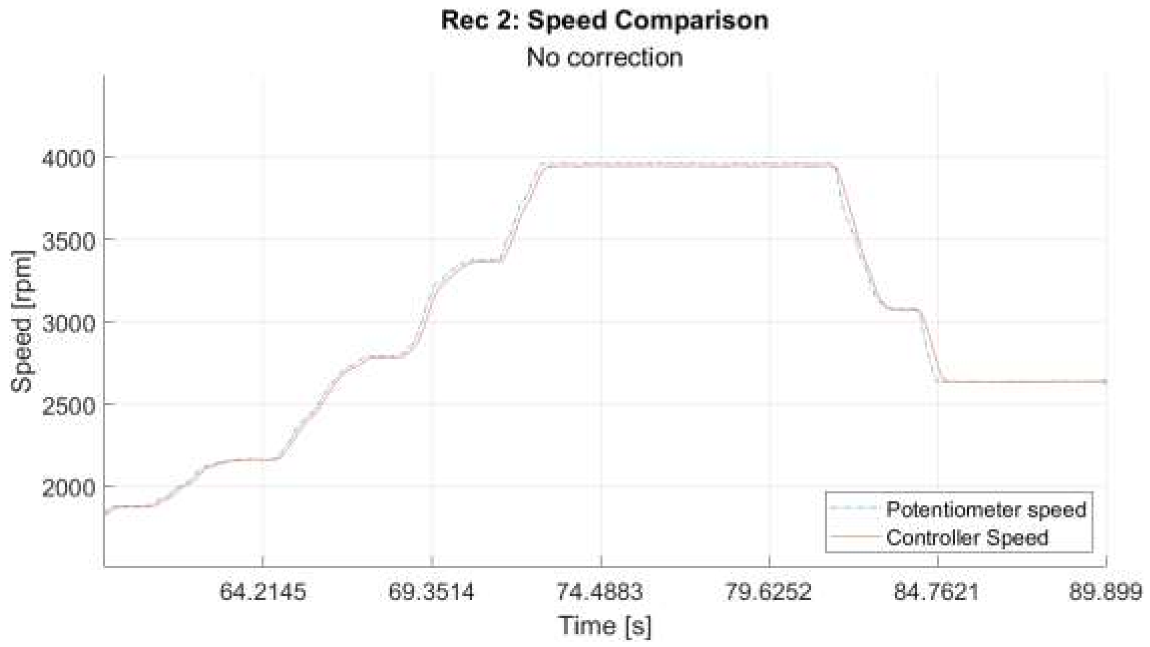


Figure 79. Test 2 Signals Details.

In order to go in deep with the difference cause, the Test 3 step verted on the controller ramp set removal. The following plots didn't count anymore on the sketch correction and on the ramp.

The performed tests were like the previous, the Recording 1 was a manual potentiometer tuning, where is possible to appreciate the quicker eMotor speed response with respect to the Test 1 in which the delay was originated from the mix of ramp and eMotor dynamics effect.

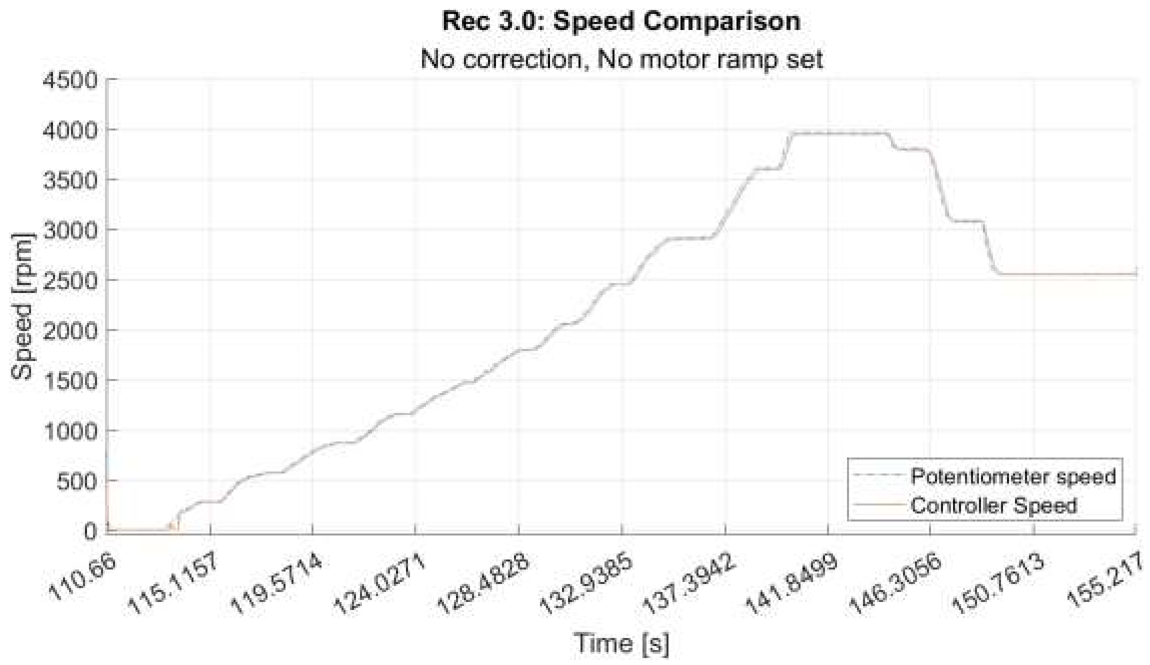


Figure 80. Test 3 Recording 1.

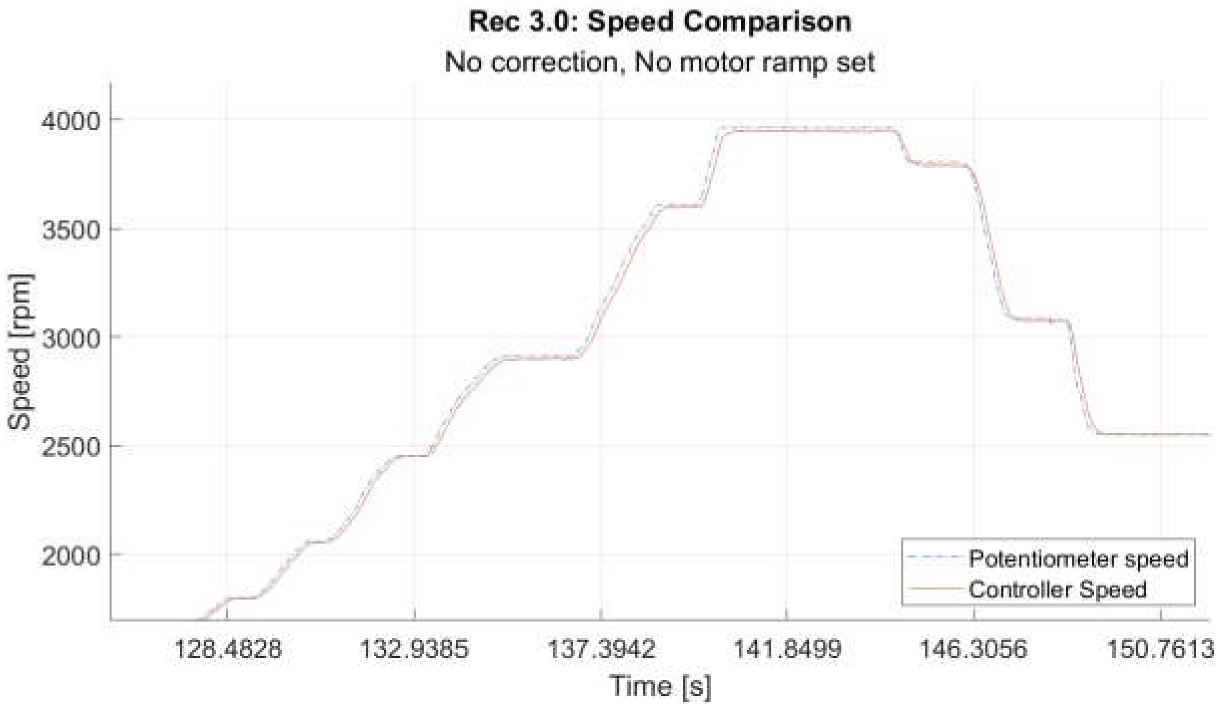


Figure 81. Test 3 Recording 1, Signals Detail.

In the Recording 2 is possible to appreciate how quick the eMotor reacts to the on/off switch. It is better to underline that the steady-state speed is not anymore the maximum admissible, but an intermediate value, despite this is still clear how steep is the response, without the need of any measurement.

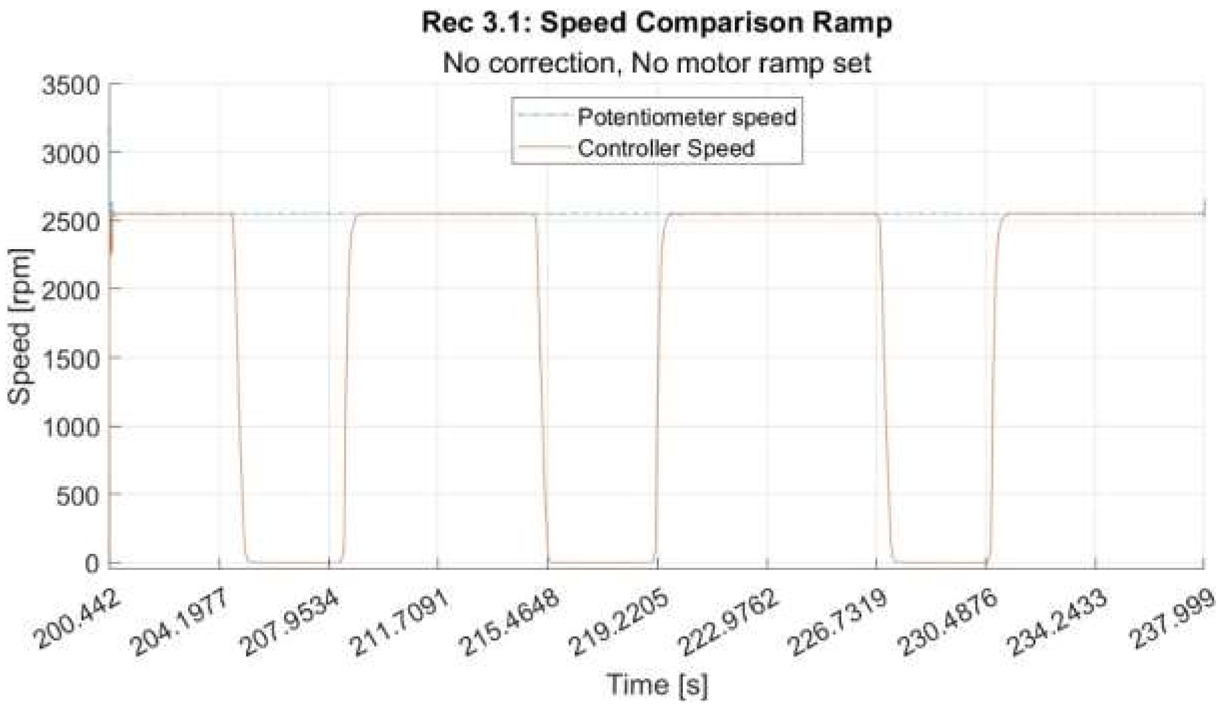


Figure 82. Test 3 Recording 2.

The last analysis performed, Test 4, followed the same procedure of the previous ones but dealt directly with the bit values, so avoiding the data loss coming from the mapping procedure; it is possible to see it into the Arduino Sketch commented rows where the bit values were sent to the Serial.

In addition to that, the bit range is one order of magnitude greater than the speed ones, allowing an enhanced precision.

The tests performed are exactly as the previous in which no code corrections or ramp are used. As in the Rec 3.1, the steady-state speed is not the maximum one, 37708 bits corresponds to 2302 rpm.

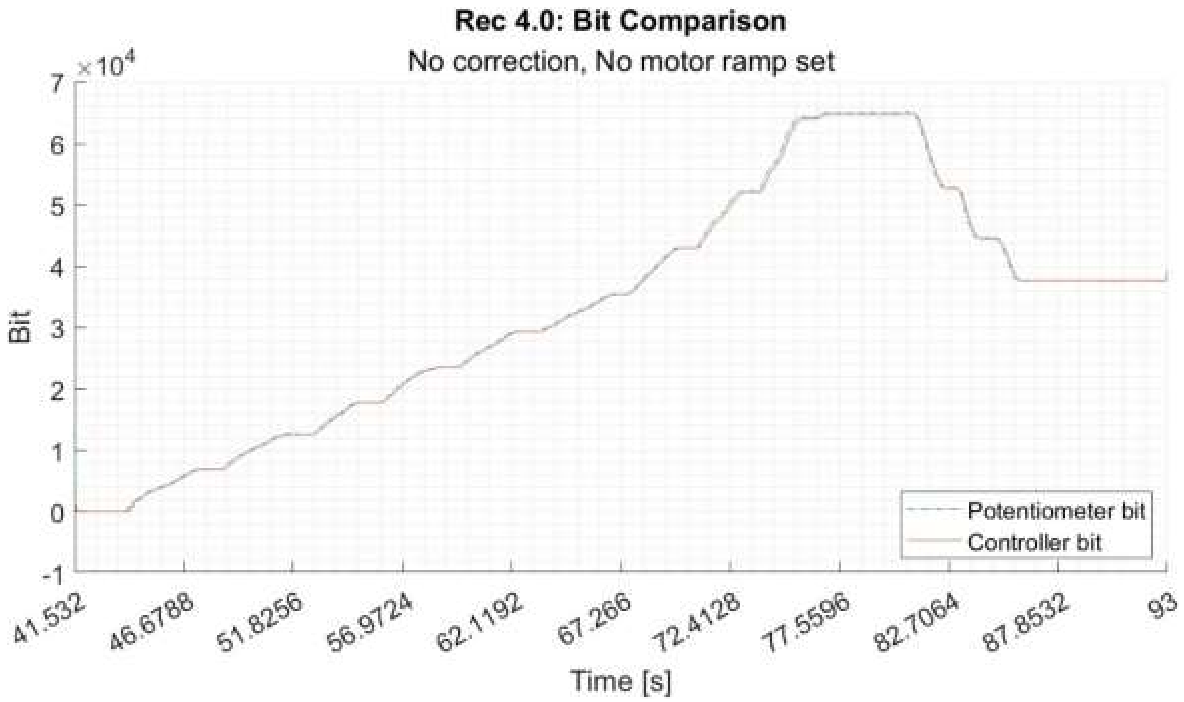


Figure 83. Test 4 Recording 1.

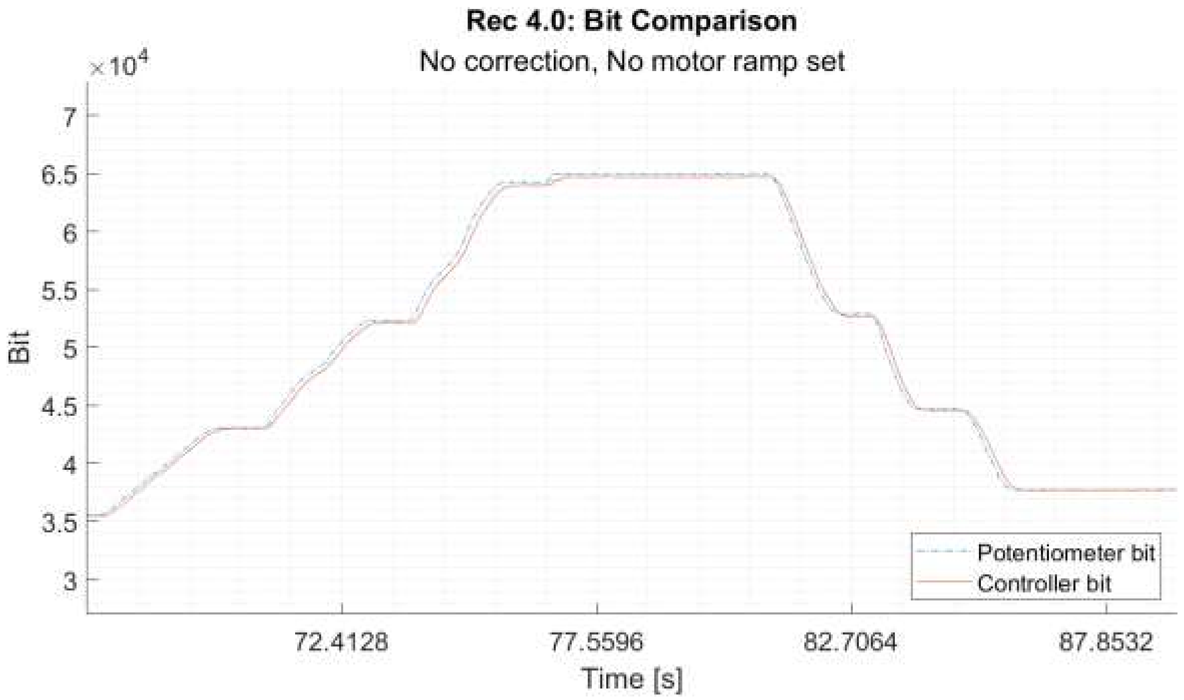


Figure 84. Test 4 Recording 1, Signals Detail.

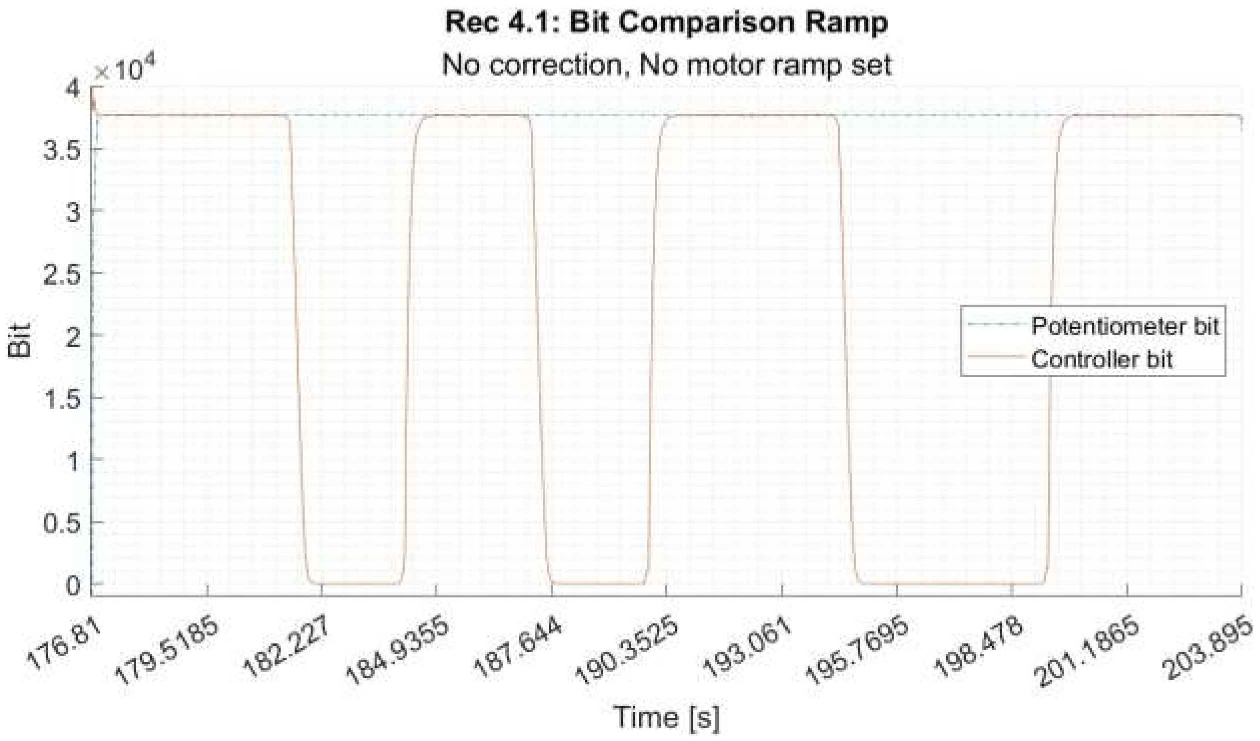


Figure 85. Test 4 Recording 2.

After conducting a thorough analysis and studying the plots, it was determined that the discrepancy observed in the signals can be attributed to the dynamics of the eMotor.

4.2.2 Controller Configuration

As in the above Case 1 – Static Test the controller must be set according to the layout implied. The best starting point is following the The configuration and interfacing of the controller with the eMotor is performed using the Escon Studio Software, developed by Maxon. The controller is connected to the software via the micro-USB port, allowing for convenient setup and configuration.

Getting Started procedure, modifying only the sections related to the control signals and feedback.

Even this time, the configuration began selecting the speed control working mode, despite what done before, the enabling signal come from a 3-position switch device, being also charged of direction of rotation selector.

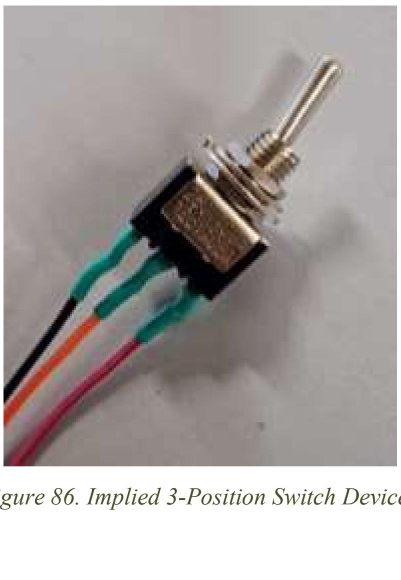
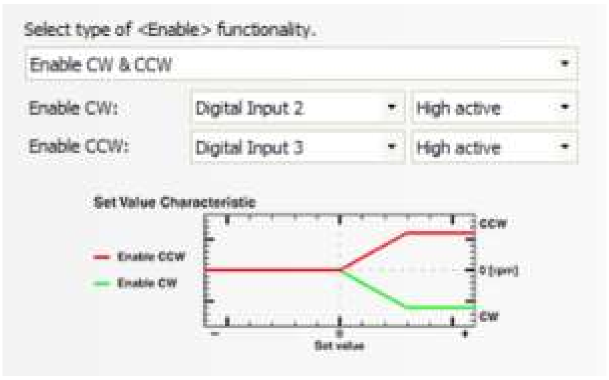


Figure 86. Implied 3-Position Switch Device.

The used switch is a digital device what according to the leverage position connects alternatively the central wire to one of the two on the sides. The positions are 3, as suggested by the name, one for the enabling of each side and a rest position in which the wires are isolated.

In the test case the needs were to energize only one side wire at a time, so the central wire was the one power supplied. The device could also be used switching among different voltage levels, connecting each side to a different level and leaving the central one free, so switching the position a variaton of voltage could be performed, not useful for what concerns the thesis.

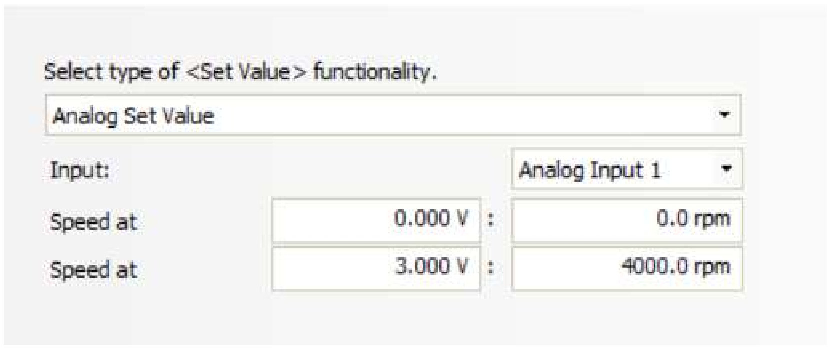
To perform this tests, the controller addresses two enabling pins, belonging to the Digital I/O J5 connector, according to the one receiving the digital HIGH signal the sense of rotation is flagged, totally fitting the switch feature. The feeding for the device was obtained by the sixth connector pin, what provides a 5V output voltage, no ground connections are needed.



Select type of <Enable> functionality.
Enable CW & CCW
Enable CW: Digital Input 2 High active
Enable CCW: Digital Input 3 High active

Figure 87. Two Pins Enabling Signal Characteristic.

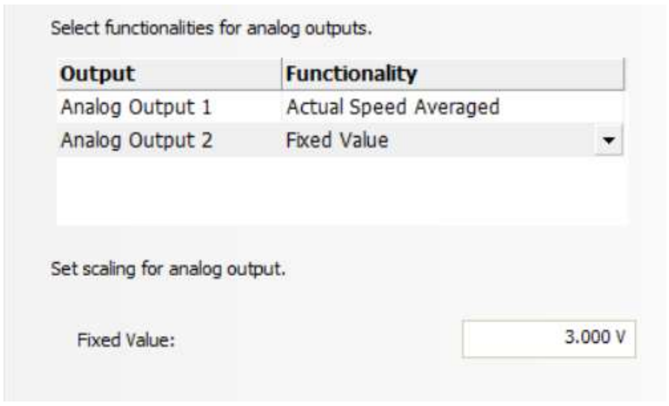
Once done that the analogic part was adjusted, the control signal was moved to the analog one and the potentiometer extreme values were merged with the speed range as shown. The 3 volts maximum come by the feeding voltage. The potentiometer signal and GND were connected, the first to the pin number one of the Analog I/Os J6 connector, the second one to both the pin number 2 and the GND port.



Select type of <Set Value> functionality.
Analog Set Value
Input: Analog Input 1
Speed at 0.000 V : 0.0 rpm
Speed at 3.000 V : 4000.0 rpm

Figure 88. Potentiometer Values Set.

The potentiometer power come from an analogical controller pin able to produce a fixed voltage level. This one was set to a 3V tension, according to the set value range.



Select functionalities for analog outputs.

Output	Functionality
Analog Output 1	Actual Speed Averaged
Analog Output 2	Fixed Value

Set scaling for analog output.

Fixed Value: 3.000 V

Figure 89. Analog Output Fixed value.

4.3. Case 3 – Wireless Control

The base control last step was to move on a wireless control. To perform this, its was decided to exploit the original controller receiver system.

The transmitter wireless signal was out of our scope since it was charged to deal with the own Spektrum SR6000T receiver, so no matter about it. The focus was on the receiver output signal since the idea was to route it to the eMotor controller. The devices designed for RC vehicles, as the aforementioned receiver, connects to typically employed servo motors by means of a manufacturer connector.

The interface consists of a series of lines, with each line serving a specific purpose. The first line is dedicated to the battery port, followed by a line designed for the throttle servo motor connection. The third line is reserved for the steering servo, while the remaining lines are left available for any auxiliary systems.

Each connector in the interface consists of three pins. The first two pins are dedicated to the ground (GND) connection and voltage feeding, respectively. The third pin carries the signal, which is of PWM type.

By utilizing the existing receiver system and designing a suitable interface, the wireless control aspect of the base control system was effectively implemented. The use of the PWM signal and the compatibility with servo motors allowed for easy management and control of the eMotor using the wireless connection.

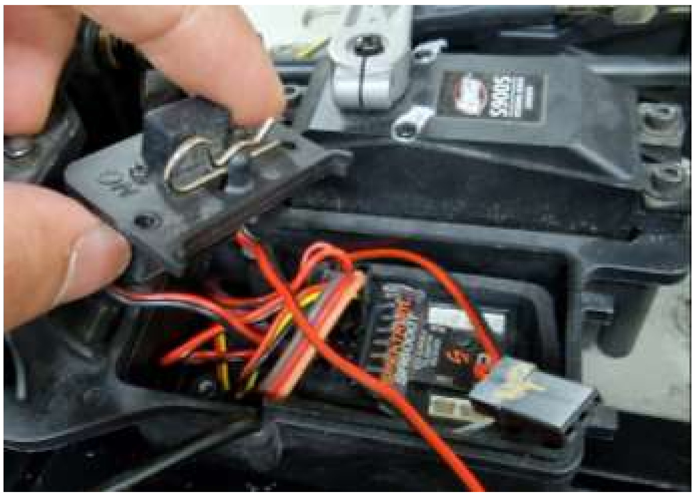


Figure 90. Spektrum SR6000T receiver and connector.

The study began by analyzing the signal from the receiver using the Escon Studio software system overview. The objective was to identify the PWM thresholds for the signal.

Through this analysis, the following duty cycle thresholds were determined: 13.6% for the throttle rest position and 19% for full throttle. These values provided important information for setting up the control system.

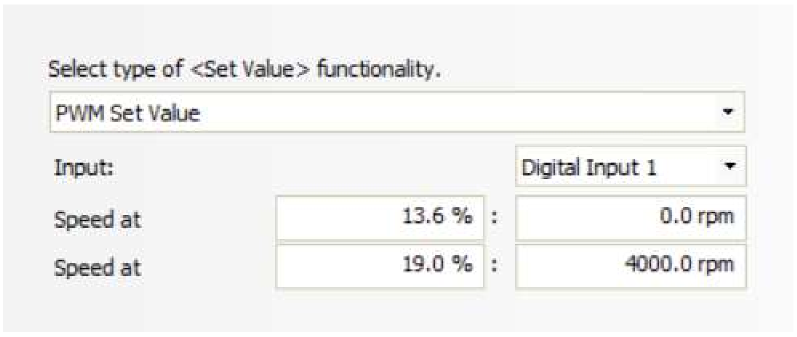
Subsequently, a proper wiring configuration was established between the receiver and the controller. In this configuration, only the signal and ground cables were utilized, as the eMotor was powered directly from the battery supply.

The decoding of the signal played a crucial role in this process, and the Speedgoat system played a significant part in this signal decoding, as described in its dedicated chapter. This involvement ensured accurate interpretation and utilization of the received signal.

With this setup and configuration, the vehicle is now capable of wireless control, allowing for the utilization of the default steering control alongside the newly implemented wireless throttle control. This enables comprehensive and efficient control over the vehicle's movements.

4.3.1 Controller Configuration

This configuration accounts for a digital PWM speed signal as in the Case 1 – Static Test. What differs are the signal thresholds, so the signal controller configuration is set accordingly.



Select type of <Set Value> functionality.

PWM Set Value

Input: Digital Input 1

Speed at 13.6 % : 0.0 rpm

Speed at 19.0 % : 4000.0 rpm

Figure 91. Controller PWM Thresholds - Wireless Trasmitter.

In this case, the switch continues to be responsible for generating the enabling signal and selecting the speed rotation, being the only wired component to deal with. The controller setup remains unchanged, including the wiring configuration.

Since there is no need for analog signals, both input and output, the corresponding connector can be left empty or completely removed from the controller. This means that the controller can operate without any physical connections to analog devices or components.

4.4. Case 4 – Speedgoat Models

In the final step of the control process, the Speedgoat device is used to achieve control over the driveline and so the vehicle. The Speedgoat device offers a wide range of control methods and interfaces, making it a suitable choice for this application. In this case, control is accomplished through a PWM signal.

The decision to employ the Speedgoat device stems from its ability to handle multiple signals and data simultaneously, anticipating future developments where the vehicle will be implied in testing purposes.

Control is performed using Simulink models, taking advantage of the diverse signal and data processing capabilities offered by the software. Specifically, three models have been developed, to enhance the analysis of vehicle signals, with each model focusing on a specific block outlined in the Simulink chapter.

In this setup, only the switch remains, as its purpose is to generate a quick enabling signal. However, it is possible to remove the switch and replace its functionality with a Simulink Switch. In such a scenario, producing a digital HIGH signal for controller enabling would be the sole requirement.

The PC is still involved in the control loop, responsible for requesting speed values. However, it is also possible to upload the code onto the Real-Time Machine and execute it in a loop, exploring alternative methods to modify speed and steering values.

Overall, the application of the Speedgoat device, combined with Simulink models, provides an efficient and flexible control solution for the vehicle, while also paving the way for future developments and testing capabilities.

Analog Read Model

The default signals are routed from the wireless receiver to the default servo motors. By studying these signals, the goal is to determine the appropriate values to generate a similar signal for controlling the steering servo. Additionally, the model aims to comprehend the signal directed to the throttle during the wireless control phase.

The configuration file, which determines the working mode, does not impact the operation of the model, as the Analog part functions in the same way regardless of the mode. Therefore, the focus is primarily on understanding the signal behaviour and identifying the values necessary for generating the desired control signals.

The model itself is relatively simple yet effective. It consists of a Setup block on the left, an Analog Input block, a couple of Scopes for visualizing the signals, and a Display to present the results. This setup allows for real-time monitoring and analysis of the PWM signals, aiding in the development of the appropriate control values for the steering servo and throttle.

The Analog Input block is adjusted to read voltage in the range of 0-5V, primarily used for visualizing the PWM signal, having a maximum voltage of 3.3V. This block outputs four signals, although only the first one is taken into account, making it capable of accommodating more complex data acquisition if needed. The Visualisation Scope is employed solely for the purpose of exhibiting the signal, while the Recording Scope enables data logging and routing of acquired data to the MATLAB Workspace for better plotting capabilities.

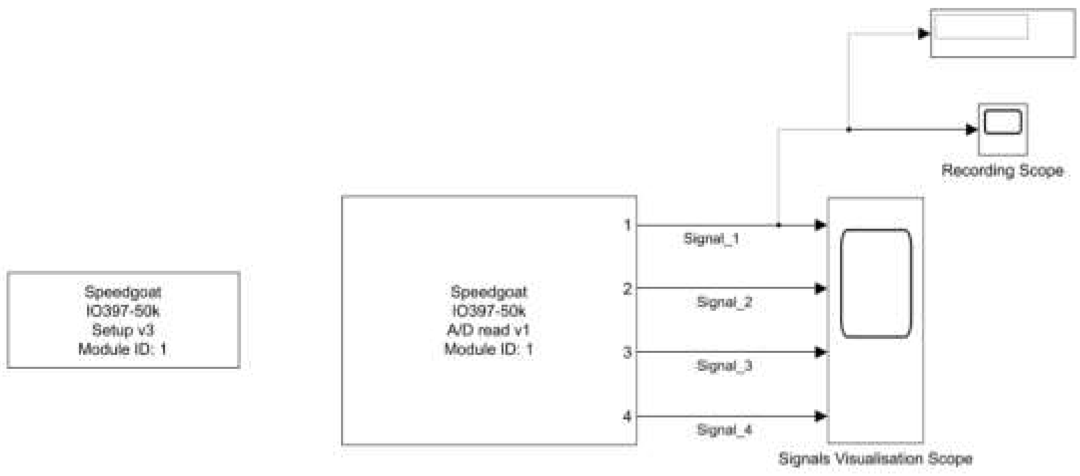


Figure 92. Analog Reading Model.

The signals read are here shown, a recording accounts for the signal directed to the throttle and the other to the steering command. The command signal is just a random movement of the proper transmitter trigger allowing a certain variation in values.

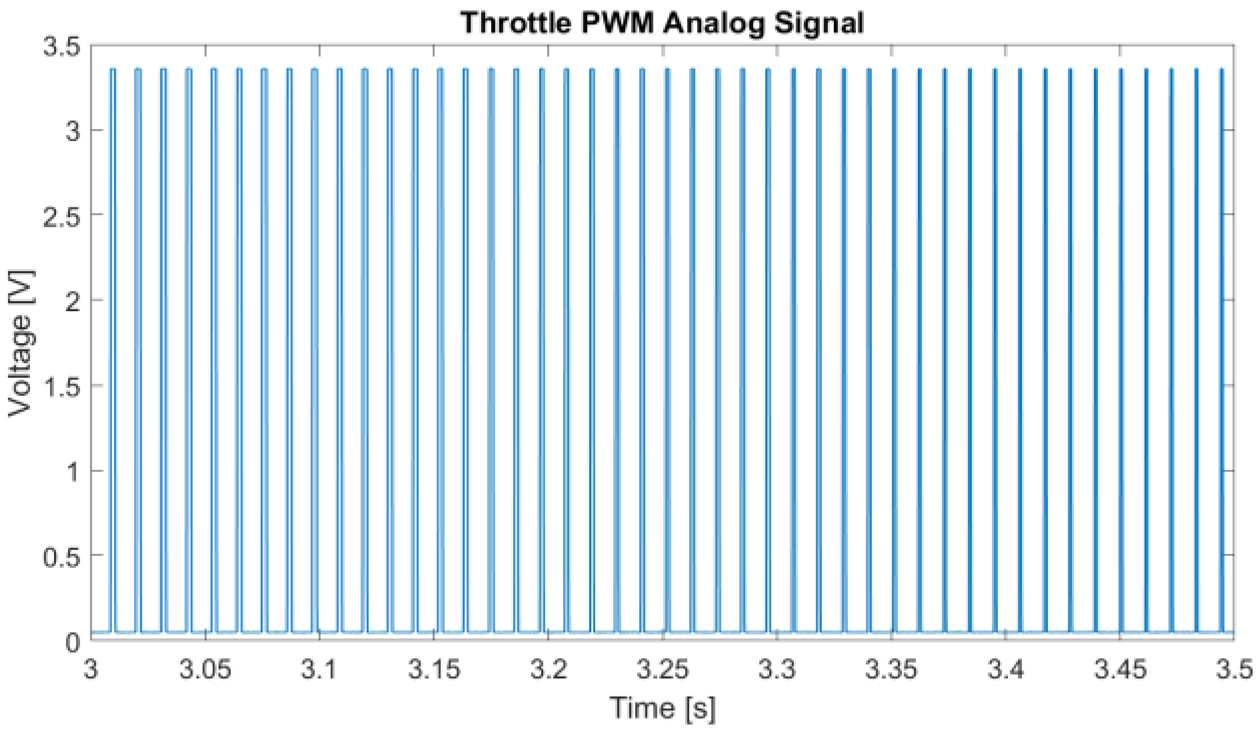


Figure 93. Throttle PWM Analog Read.

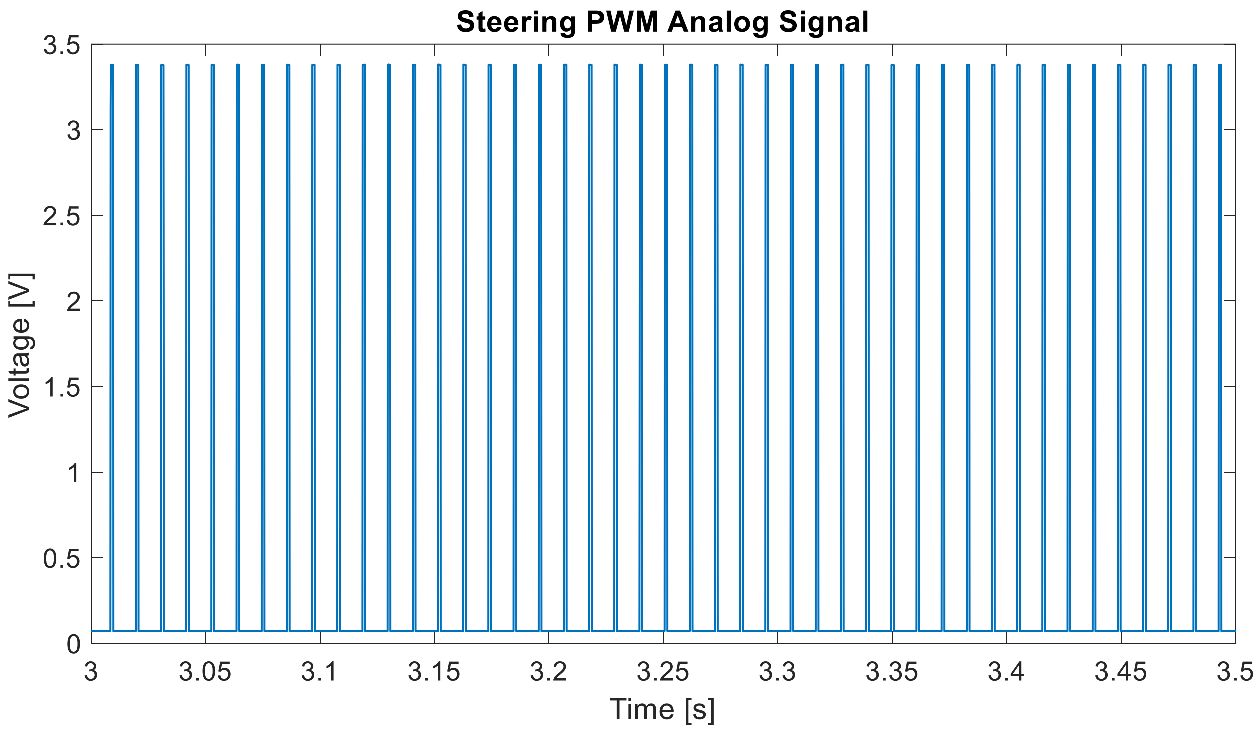


Figure 94. Steering PWM Analog Read.

PWM Capture Model

The Signals Analog Reading Model is only useful for the visualisation of digital PWMs, understanding the correct working. This further model aims to discover the signal duty cycle thresholds, relevant for controlling purposes. Following the previous model those are found for throttle and steering PWM signals coming from the receiver.

The model provides various outputs, but the most relevant one is the duty cycle. The duty cycle is represented as a value ranging from 0 to 1, indicating the percentage of time the signal is in the HIGH state. Additionally, the model outputs the period, although it may not be of immediate interest. The Escon Controller used in this case is capable of handling a wide frequency range, and the period represents the number of ticks between two triggering events, with a HIGH transition set as the triggering event. The tick duration is 13.33 ns, which can be obtained from the setup block. By combining the tick duration with the number of ticks, it is possible to determine the PWM frequency.

The model incorporates the setup block, which must account for the HIL (Hardware-in-the-Loop) configuration file. Additionally, it includes a Capture Block to capture the analog signals, a Scope for visualization, and another Scope for data saving, enabling further data processing. Finally, a Display block is included for each signal to provide relevant information.

By utilizing this model, the duty cycle thresholds of the analog signals can be identified, enabling precise control of the throttle and steering signals. The captured data can be further processed and analysed for fine-tuning the control system.

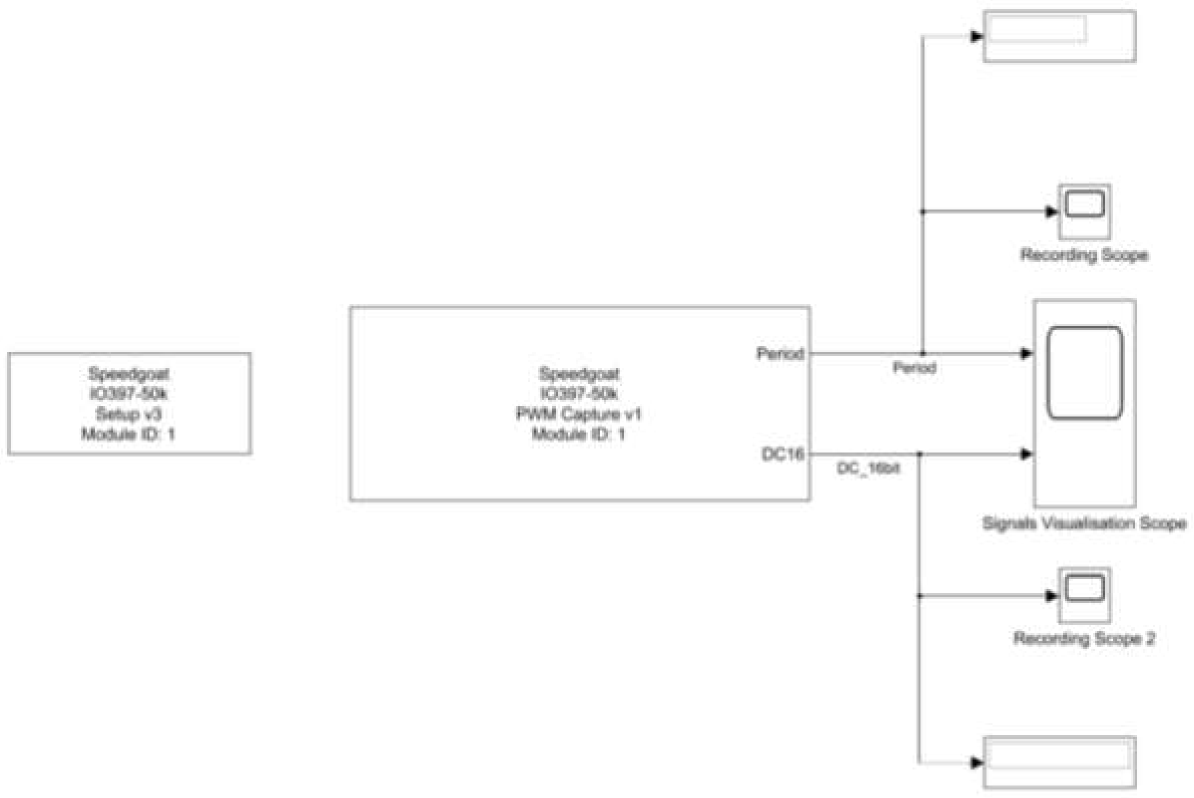


Figure 95. PWM Capture Model.

The acquired Duty Cycles are displayed in the plots, with the command being a random triggers movement.

In the first plot, the duty cycle directed to the throttle servo motor is depicted. This signal is used to control the ePowertrain in the application. The duty cycle ranges from 8.4% to 19%. It is important to note that the motor controller is unable to read duty cycles lower than 10%. However, it is worth mentioning that the signal below the trigger rest position, at 13.6% duty cycle, is used to control the vehicle driveline brake system. In the electrified version of the vehicle, this brake system is not utilized, as the eMotor functions as a brake starting from the rest threshold according to the controller settings. Therefore, the exploitable range for controlling the ePowertrain lies within the 13.6% to 19% duty cycle range.

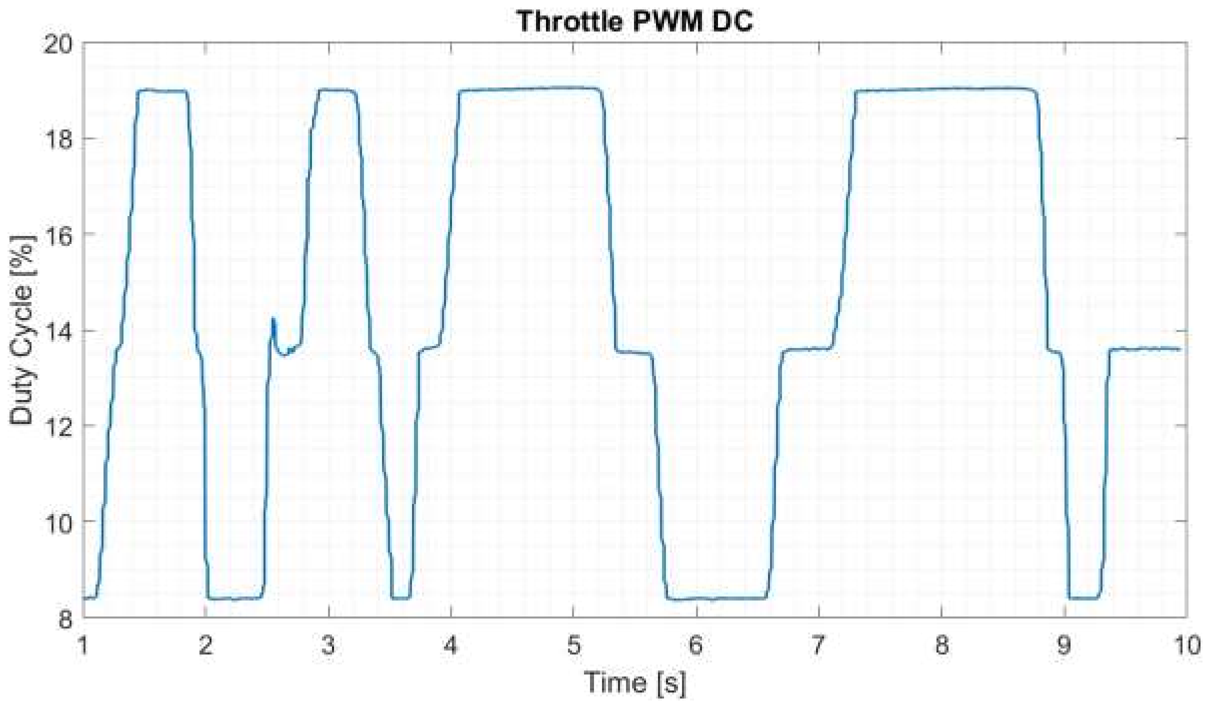


Figure 96. Receiver Throttle PWM DC.

To obtain a more detailed and refined plot, an analog reading of the signal was performed and combined with the duty cycle information. This combined representation provides a better visualization, especially considering that the variation in duty cycle is not very wide.

By incorporating the analog reading of the signal alongside the duty cycle, a more comprehensive view of the signal behaviour is achieved. This combined approach allows for a better understanding of how the signal corresponds to the actual analog values.

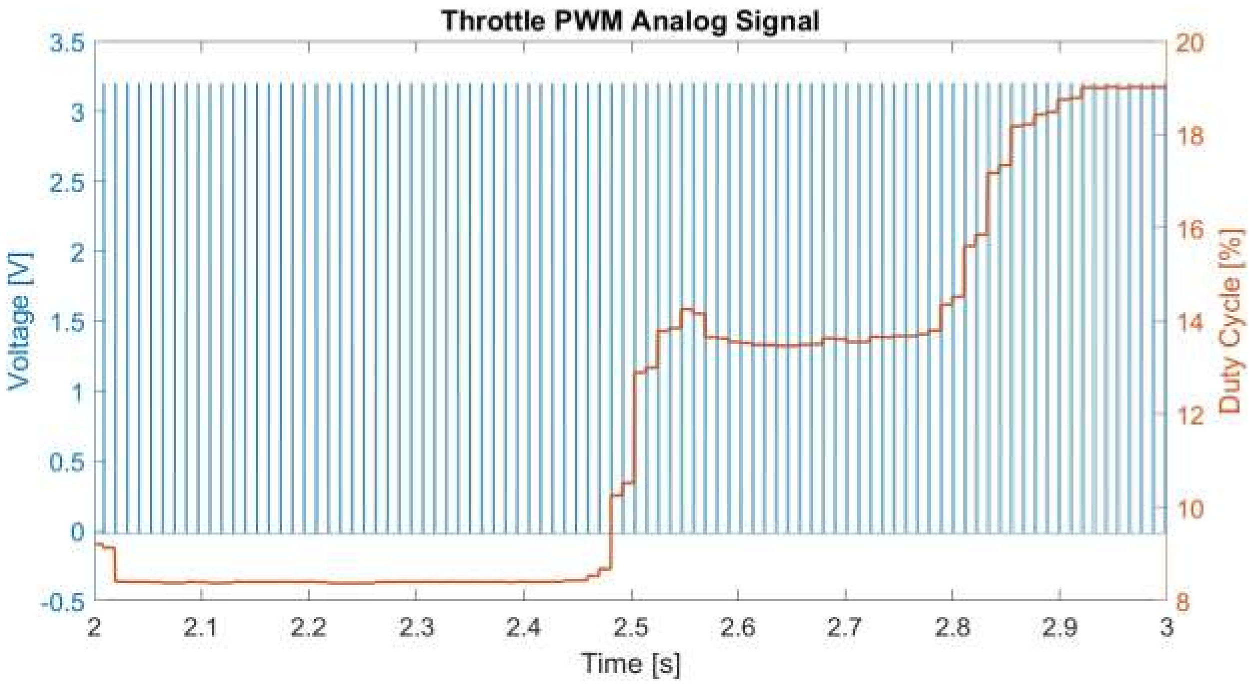


Figure 97. Throttle PWM Analog and DC.

The DC is also visualised for the steering PWM signal, the unique difference between this and the throttle one is related to the thresholds, apart from the rest position still site at 13.6%. the signal moves from 10% to 17.2%, below the rest position the steering moves the wheels to the right, over it, to the left.

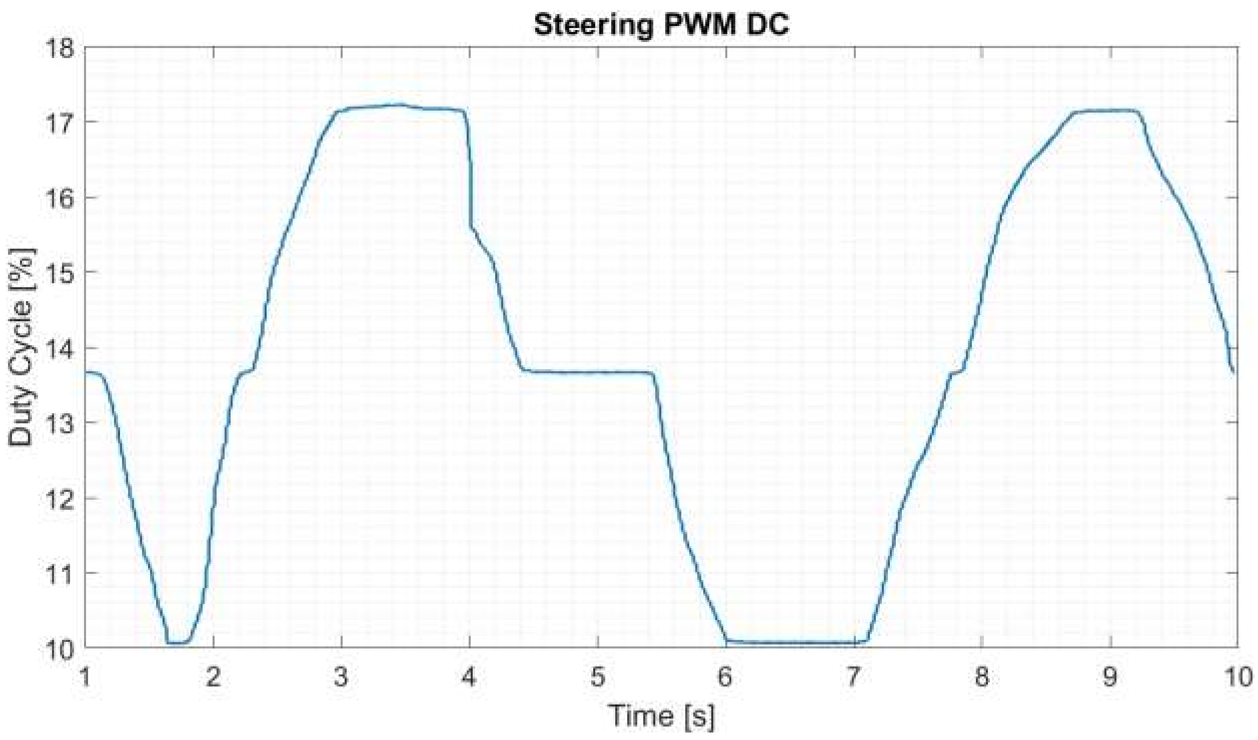


Figure 98. Receiver Steering PWM DC.

The same detailed plot is printed, just as an example of what is possible to obtain from the system.

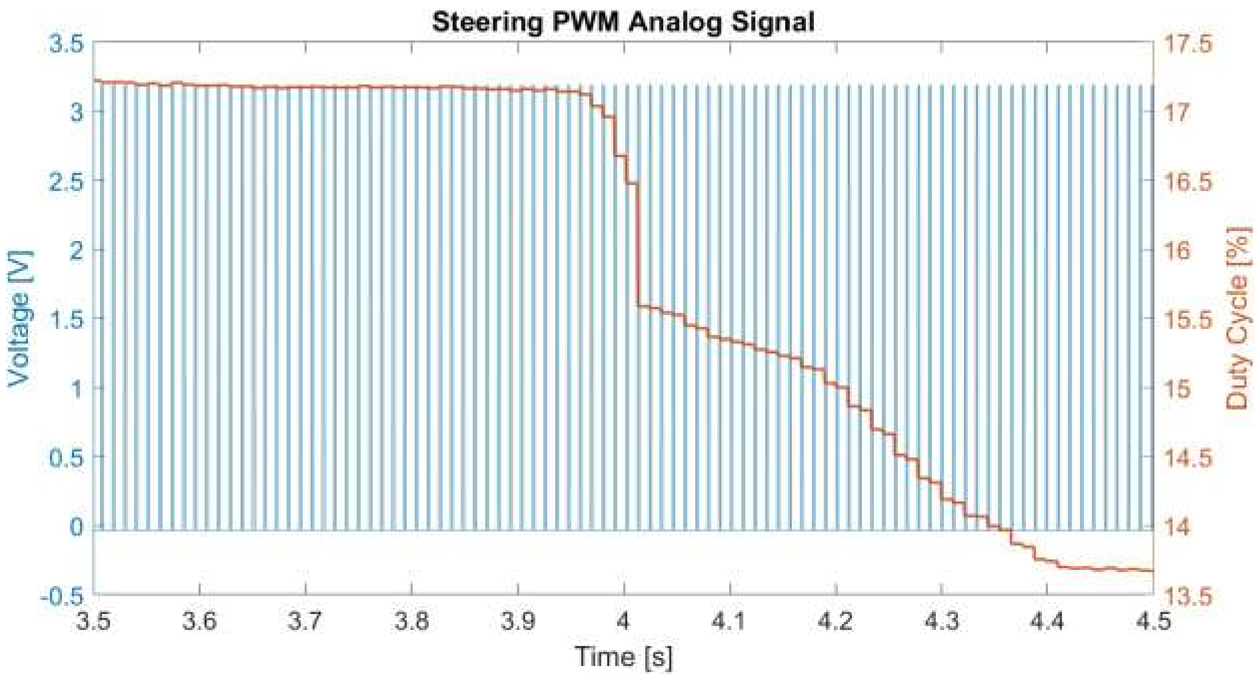


Figure 99. Steering PWM Analog and DC.

Lastly, the signals frequency is shown, even if a bit of noise is present, this is constant at about 91 Hz.

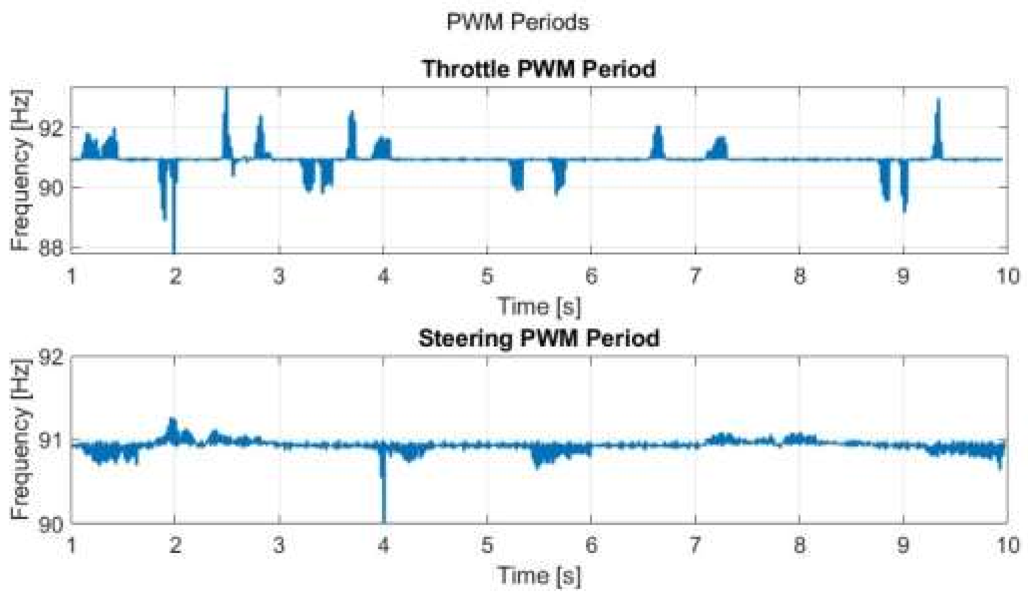


Figure 100. PWM Periods.

PWM Write Model

The final stage of the control process focuses on controlling the eMotor by sending a PWM signal. A similar approach could be applied to the steering control by appropriately setting the thresholds. However, in this case, the focus is on throttle control.

To avoid continuous resetting of the controller parameters, the decision is made to work with duty cycle thresholds tuned to the values received from the receiver, spanning from 13.6% to 19%. These thresholds ensure a suitable range for controlling the eMotor.

For this stage, the bitstream file "RCP" needs to be selected, enabling the production of PWM signals through the PWM Generation Block. The model incorporates additional blocks compared to the previous one. The Setup Block is present, along with scopes and displays, fulfilling the same role of data saving and visualization.

The new blocks added implies the PWM Generation block, a vertical slider and a look-up table. The PWM Generation block, symmetrically to the Capture one, allows to switch among different parameters. In this case, the inputs to the block are the duty cycle and the period. The period is obtained from the capture model, while the duty cycle is controlled by the vertical slider, enabling real-time speed regulation.

The look-up table is just used for a sake of directness of the data visualization, it moves the Duty Cycle value to the corresponding speed.

Overall, this model represents a powerful tool for complete control over the system. It facilitates the generation and adjustment of PWM signals for controlling the eMotor, allowing for real-time speed regulation and providing a visual representation of the corresponding speed values.

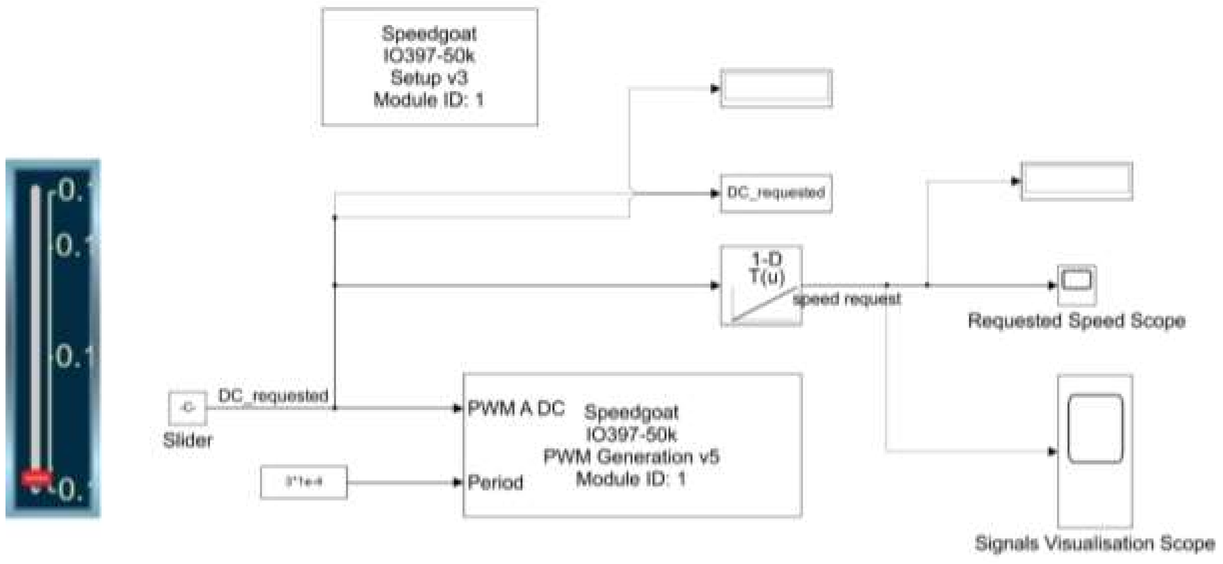


Figure 101. PWM Write Model.

Conclusions

In conclusion, this research work focused on the electrification and control of a 1:5 scale off-road racing vehicle, aiming to achieve a vehicle ready for dynamic tests with precise control capabilities, overcoming the limitations of the stock transmitter/receiver system and leveraging the advantages of an electric powertrain, smoothing the ICE vibrations, allowing to exploit sensors directly to the vehicle chassis for a better data acquisition. The replacement of the 2-stroke internal combustion engine with a brushless DC motor, planetary gearbox, and electric motor controller resulted in a precisely controllable vehicle.

In this configuration the vehicle could be well employed in lateral dynamic tests, like the constant cornering radius, exploring its behaviour by varying the suspension setup and the weight distribution, but not only. The high torque available let it perfectly suitable for longitudinal dynamics study, it could be used to investigate the maximum slope achievable, varying the friction coefficient and adding weight in different locations to well depict the vehicle response.

The thesis systematically explores different aspects of the project, starting with the selection of ePowertrain components. Each component is chosen after careful evaluation, considering the specific reasons behind its selection, in order to achieve the best possible replacement.

Subsequently, a comprehensive examination of the utilized hardware is presented, providing a detailed analysis of each device, with a specific focus on the main key features, employed software, implementation details and setup performed. The first device discussed is the Arduino Portenta H7, a professional version of Arduino boards, which was selected in the early stages of the project to explore control principles. The Arduino IDE software is used, and custom sketches are developed to serve various purposes, including platform analysis, system control, and precise signal recording to identify mismatch among measured values. A similar level of detail is provided for the Motor Controller, a crucial component that performs essential tasks and requires optimal configuration, including the wirings followed by the different test cases. Additionally, the integration of the sophisticated Real-Time Target Machine, Speedgoat, is explained. This system sets itself apart from the previous hardware components due to its versatility and adaptability, particularly its operating mode by means of Simulink real-time models. This integration offers extensive opportunities for precise parameter adjustment, fine-tuning, and real-time data acquisition, ensuring the alignment between commanded signals and acquired data.

The Components Installation chapter provides a detailed overview of the engine swap process, covering essential aspects such as the removal of unnecessary components, various connection attempts, the design and needs of CAD models, and the exploration of solutions for effective power transmission from the eMotor to the ground. These meticulous steps were fundamental in achieving a seamless integration of the electrified powertrain components, resulting in optimal performance and driveline efficiency, thereby enhancing the correct response of the vehicle.

The last chapter provides an in-depth explanation of the conducted Experimental tests, where the focus was on achieving optimization and control. It covers the implementation of additional devices such as potentiometers and switches, elaborating the tasks they performed. The hardware setup for each test, with particular emphasis on the controller as the key component in the driveline, is thoroughly discussed, including the specific parameters set. The obtained results are presented, along with the implemented measures to improve the response and ensure reliable outcomes. The chapter highlights the initial control attempt using the Portenta Board during the bench testing phase, followed by the transition to potentiometer analog control. Being some discrepancy in requested and obtained speed born, the causes are carefully examined, acquiring data using Arduino and visualising them through MATLAB, facilitating more efficient data filtering and management. Additionally, the chapter explores the wireless control implemented on the vehicle by means of the stock transmitter/receiver system, allowing it to move freely within the testing environment. Finally, the complex control method by means of Speedgoat is elucidated, providing insights into the construction of the required Simulink Real-Time model for seamless integration and operation, together with acquired data plot to show the potentiality of it for future testing implementation.

The created vehicle well adapts to further developments going in deep with the main topics of vehicle dynamics and system control.

The electrified Losi 5ive-T provides a versatile platform to compare and evaluate different control methods in terms of accuracy and reliability, exploiting advanced control method with refined systems. This is facilitated by the efficient management of control signals by means of the Maxon Escon Controller, being able to work in a huge voltage range, for digital and analogical signals.

Another potential future development is the integration of a predefined track for autonomous navigation. This would allow the electrified Losi 5ive-T to autonomously follow a predetermined path, eliminating the influence of the user. This advancement would enable the vehicle to gather vehicle response data, on the same pathway, by varying the setup, such as weight distribution or suspension characteristics. This development opens the possibility for implementing an autonomous driving configuration. To achieve this, the vehicle must be equipped with sensors such as cameras, radars, or other suitable solutions to enhance perception and ensure the navigation into a non-predefined track.

Another promising idea for future exploration is vehicle wireless control, which would provide increased freedom of motion. This capability would allow the vehicle to execute non-predefined patterns by means of user control, enabling the acquisition of several data and expanding the range of control possibilities. The implementation of an offline acquisition system would complement wireless control, facilitating comprehensive data analysis and further improving the understanding of vehicle behaviour.

Overall, this research work contributes to the field of vehicle electrification and control, demonstrating the successful transformation of the off-road racing vehicle and highlighting the advantages achieved in terms of precise control and vibration smoothness.

Bibliography

[1] R. M. Freeware, «Roger Meier's Softwares,» [Online]. Available: https://freeware.the-meiers.org/.

[2] Maxon Motor AG, «Technical Support - Escon Product Line,» Maxon Motor AG, [Online]. Available: https://support.maxongroup.com/hc/en-us/sections/360001171953-ESCON-product-line.

[3] RS Components, «Motore c.c. Brushless RS PRO, 24 V c.c., 4000 giri/min,» [Online]. Available: https://it.rs-online.com/web/p/motori-cc/5366030.

[4] Maxon Motor AG, «Maxon Products - Controller,» Maxon Motor, [Online]. Available: https://www.maxongroup.com/maxon/view/product/control/4-Q-Servokontroller/409510.

[5] Maxon Motor AG, «Maxon Group @maxonglobal,» [Online]. Available: https://www.youtube.com/@maxonglobal.

[6] Maxon Motor AG, «Maxon Escon 50/5 - video overview,» [Online]. Available: https://vimeo.com/53416940/71e6be94a3?embedded=false&source=video_title&owner=12583329.

[7] Losi, «Losi Product,» [Online]. Available: https://www.losi.com/.

[8] Maxon Motor AG, «IxR Compensation,» [Online]. Available: https://support.maxongroup.com/hc/en-us/articles/360006040894-ESCON-Speed-controller-open-loop-with-IxR-Compensation.

[9] Maxon Motor AG, «ESCON 50/5, 4-Q Servocontroller for DC/EC motors, 5/15 A, 10 - 50 VDC - Overview,» [Online]. Available: https://www.maxongroup.com/maxon/view/product/control/4-Q-Servokontroller/409510?etcc_cu=onsite&etcc_med=Header%20Suche&etcc_cmp=mit%20Ergebnis&etcc_ctv=Layer&query=escon%2050#vimeovideo53314343.

[10] Maxon Motor AG, «Escon 50/5 Hardware Reference,» Brünigstrasse 220 P.O.Box 263 CH-6072 Sachseln, 2013.

[11] Arduino, «Portenta H7,» [Online]. Available: https://store.arduino.cc/products/portenta-h7.

[12] Arduino, «Arduino® Portenta H7 Collective Datasheet,» 2022.

[13] Zenoah, «Zenoah G320RC,» [Online]. Available: https://www.zenoah.com/int/products/hobby-engines/g320rc/967289001/.

[14] «Wikipedia,» [Online]. Available: https://en.wikipedia.org/.

[15] Speedgoat GmbH, «Speedgoat,» [Online]. Available: peedgoat GmbH.

[16] Speedgoat, «IO397 Specifications,» [Online]. Available: https://www.speedgoat.com/products/simulink-programmable-fpgas-fpga-i-o-modules-io397.

[17] F. Pesacane, "Numerical models and experimental measurements for parametric dynamic analyses on an RC vehicle in scale 1:5," 2019/2020.

[18] Wikipedia, «Dinamica del Veicolo,» [Online]. Available: https://it.wikipedia.org/wiki/Dinamica_del_veicolo.