POLITECNICO DI TORINO

Master's Degree in Automotive Engineering

Master's Thesis

Design of autonomous control systems for a Driverless race car: Remote Emergency System and Autonomous State Machine

Supervisor:

Prof. Nicola Amati

Co-supervisors:

Prof. Andrea Tonoli

Prof. Angelo Bonfitto

Dott. Stefano Feraco

Candidate

Gennaro Sorrentino

Academic Year 2019/2020

Abstract

Autonomous driving is the most challenging research field in automotive in recent years, alongside emissions reduction. Particularly, the progressive reduction of the human driver's contribution in the vehicle control loop, started with the first milestones in the active safety field as ABS and ESP and with the aim of completely getting rid of the human driving, is increasing constantly the road safety eliminating the human error.

Since 2017, Formula Student competition is including a Driverless class, too, in order to stimulate students and future engineers to face the design and realization of a Driverless race car prototype. The specific rules of this class regulate the design and implementation of the crucial subsystems for a driverless vehicle: the perception, the actuators, and the safety components.

This thesis work is a contribution to the development of the Driverless prototype by Politecnico di Torino. More specifically, the first part concentrates on the communication and testing of the Remote Emergency System (RES), which is a radio component useful to give remote commands to the vehicle. It is really important since it can activate the Emergency Braking System (EBS) if the team remotely notices issues during the autonomous mission. The second part focuses on the design of the Autonomous System state machine, which is the upper supervisor of the vehicle decision-making part because, on the basis of the inputs from sensors and remote commands, it is in charge of activating the actuators and the specific control algorithm for the selected mission.

The work followed a step-by-step methodology, starting from the theoretical bases and the simplest model, and adding improvements just once the previous updates were verified. However, the work on RES started directly with physical testing to rebuild its behaviour using software as Busmaster, Simulink Vehicle Network Toolbox, and Vector CANdb++. Concerning the state machine instead, the design is all virtual in MATLAB and Simulink Stateflow environments, giving physical meaning to the variables in order to simplify the subsequent implementation in the vehicle control unit.

Acknowledgements

I would like to sincerely thank my thesis tutor Prof. Nicola Amati for offering me the opportunity to work in the innovative field of the autonomous driving, and especially for the continuous interest in making it physical. This has been a push to constantly improve and grow both personally and professionally.

A special thanks also to Prof. Andrea Tonoli and Prof. Angelo Bonfitto for their suggestions and availability.

I would like to thank all the Driverless team: Stefano, Sara, Salvatore, Irfan, Luca, Raffaele, and Eugenio. Despite the pandemic, we have been capable of work as a team, and seeing the results of our work in the LIM have made me very proud.

I am grateful to my parents, who supported my decision to move in Turin for my master's degree, and to my sisters, who are always willing to listen and give advices to my doubts.

The deepest thank to Dalila for being the other half of my person and giving me the motive to keep pushing.

Finally, I am thankful also to my brother Marco to be my first supporter, and to all my friends in Bari and in Turin for all the fun and the company in these years.

Contents

List of Figures

Abbreviations

ABS = Anti-lock Braking System

ADAS = Advanced Driver Assistance System

AMI = Autonomous Mission Indicator

APPS = Accelerator Pedal Position Sensor

AS = Autonomous System

ASMS = Autonomous System Master Switch

ASSI = Autonomous System Status Indicator

EBS = Emergency Braking System

ESP = Electronic Stability Program

LVMS = Low Voltage Master Switch

LVS = Low Voltage System

RES = Remote Emergency System

R2D = Ready-to-Drive

SA = Steering Actuator

SB = Service Brake

SDC = Shutdown Circuit

TS = Traction System

TSMS/HVMS = Traction System Master Switch / High Voltage Master Switch

Chapter 1 – Introduction

1.1 Background

Vehicles are the most important mean of ground transportation for moving people and goods. From ACI data, 52 401 299 vehicles were running in Italy in 2019; in particular, automobiles were 39 545 232 units [1]. The very first models of automobiles were invented and built in the late 1800s during the Second Industrial Revolution, children of the new studies in the fossil fuels [2]. They have always been a concentration of the most innovative technologies since in a relatively small volume there is the generation of power, the exchange of forces with the ground, the control of the movement, and the accommodation for people and luggage. For example, in 1876 Nikolas August Otto invented the first car equipped with a four-stroke internal combustion engine, which was optimized for gasoline feeding and became the most adopted power unit during the First World War. Moreover, the automotive industry has led for years the revolutions in the production field; for instance in 1908, Ford Model T was the first vehicle manufactured through an assembly line and its production in series standardized this kind of production process in 1913 [3].

Nowadays, alongside emissions control, the automotive research is focused on safety both for passengers and the other road users. With 1.4 million people dying of road injuries in 2016, the World Health Organization listed road injuries in the top ten of global causes of death [4]. From ISTAT data, 172 183 accidents with injuries happened in Italy in 2019. The first three causes of crashes are distraction while driving (15.1% of accidents), lack of respect of the right of way (13.8%), and high speed (9.3%) [5]. Vehicle safety is usually classified into three big branches: *preventive safety*, concerning the ability of the vehicle to keep the driver updated on corrective manoeuvres to be taken, *passive safety*, which involves the restrain systems and the energy absorption by deformations of the vehicle body when crash is unavoidable, and *active safety*, regarding systems and design criteria allowing the driver to avoid the collision [6]. As the previous numbers state, human factor is the main criticality, so *active safety* is the most challenging safety area today. The Anti-lock Braking System (ABS) and the Electronic Stability Program (ESP) are the first milestones in the *active safety*, which evolution is now represented by *ADAS (Advanced Driver Assistance Systems)* such as the Adaptive Cruise Control and the Lane Keeping System. The future seems to be the transition towards *Autonomous Driving (AD)* which would revolutionise the way of

travelling since people would be just passengers in a computer-controlled vehicle that will ensure safety controlling the speed and continuously monitoring in real time the urban environment.

1.2 Autonomous driving: historical milestones

The idea of a *self-controlled vehicle* was already developed in the Middle Ages. As a matter of fact, some sketches demonstrate that also Leonardo da Vinci was roughly planning this possibility. Later, many scientific fictious novels imagined a world with robotized vehicles. The first prototypes of self-driving cars appeared in the 1920s, but they heavily relied on specific external inputs; for example, the car behind was controlling the protype, known as "Phantom Auto" [7].

In 1939 at the General Motors Futurama exhibit during the World's Fair in New York, there was the promise of an automated highway system for the United States revolutionising the transport of people and freight. In fact the advantages for the industrial world would be in terms of productivity, reducing the time spent in a vehicle, road safety, costs of congestion, energy consumption, and pollution. From 1980 to 2003, automotive university research focused both on "dumb" vehicles needing the guide of the surrounding infrastructures and on autonomous vehicle independent from the road. From 2003 to 2007, the U.S. Defense Advanced Research Projects Agency (DARPA) launched three "Grand Challenges" for driverless vehicles that enhanced the autonomous technology [8]. In 2004, the first challenge was hold in the Mojave Desert on a 150 miles track, but the best vehicle just covered 7 miles. In 2005, 5 vehicles over 23 succeeded in covering, and the winner of the second challenge was the Stanford University because their vehicle completed the track in 7 hours. Finally, the 2007 challenge was hold in aeronautical camp in California simulating an urban environment, and the challenge was to complete the 60 miles track in less than 6 hours. Carnegie Mellon University won [9].

More recently, private companies have advanced in autonomous driving. Google Driverless Car initiative has developed and tested a fleet of cars and started campaigns to demonstrate the potentialities of the technology. For example, it can offer mobility to the blind. In 2013, Audi and Toyota both unveiled their research programs at the annual International Consumer Electronics Show in Las Vegas. Nissan has also recently announced plans to sell autonomous cars by 2020 [8]. The most famous revolutionary car company is Tesla by Elon Musk which

is developing self-propelled vehicles really fast thanks to the vehicle ability of learning from the other vehicles of the family while working together. In particular, they send each other signals processed by their sensors, exchanging information about changing lanes and obstacle detection. So, vehicles can improve day by day [7].

Other milestones in the autonomous driving history can be seen in Figure 1.

Figure 1. Autonomous driving milestones [7]

Many advantages about autonomous vehicles can be listed:

- Reduction of crashes, eliminating the human factor, particularly due to imprudence and alcohol effect. This leads also to an overall social welfare benefit
- Mobility for those who are currently unable or unwilling to drive, leading to better independence, access to essential services, and reduction of social isolation

- Reduction of traffic congestion and its cost, since vehicles will adjust their path knowing the traffic conditions and people could however undertake other activities while in the vehicle
- Land use, since people could live further from the city cores avoiding to waste time in driving. This could revolutionise also the cities layout to a more dispersed and low-density architecture without the need of human-driver centred infrastructures
- Decrease of emissions and better energy use, thanks to a smoother driving style and a lighter vehicle structure, consequence of the reduced crash probability [8].

1.3 Autonomous driving: SAE levels and layout

The first classification in the automotive field is between *automated vehicles* and *autonomous vehicles*. The former class includes all the improvements useful to make "more automatic" vehicles, in particular concerning the driver assistance (ADAS). The latter describes the last step in the automation. More specifically, Regulation 2019/2144 of the European Parliament and of the Council of 27 November 2019 on type-approval requirements for motor vehicles defines *automated vehicle* and *fully automated vehicle* based on their autonomous capacity: an *automated vehicle* is a vehicle designed to be conducted by a human driver which must be the supervisor also when the vehicle takes itself the control in certain period; an *autonomous vehicle*, instead, is directly designed to get the driver out from the control loop in all the traffic situations [10] [12].

SAE (Society of Automotive Engineers) has standardized the "Levels of Driving Automation" in standard J3016 (Figure 2) [11].

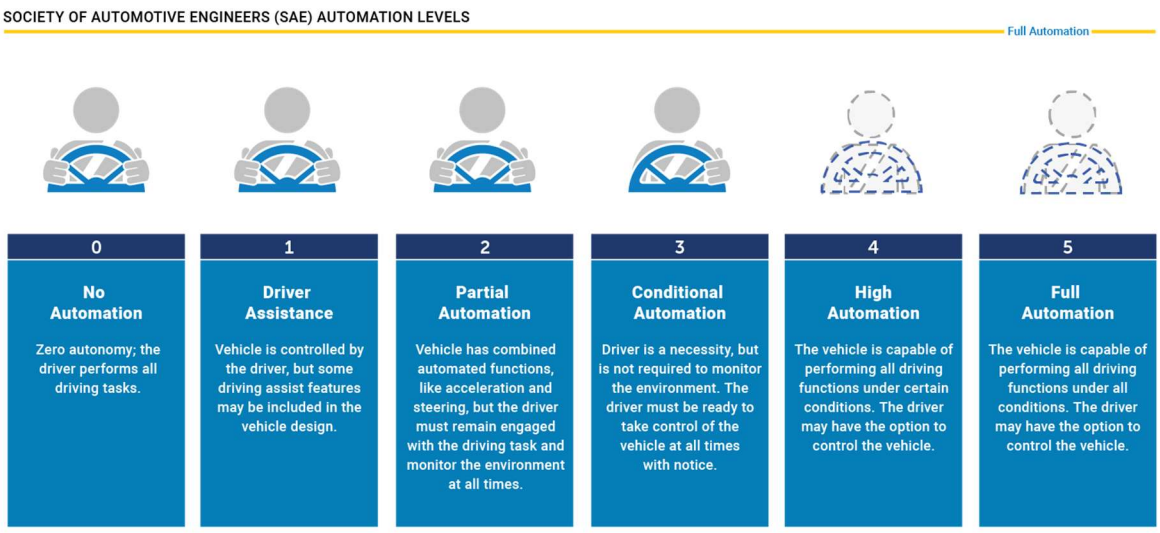


Figure 2. SAE J3016 Levels of Driving Automation [11]

The first three levels require the mandatory supervision of the human driver regarding steering, braking, and accelerating as needed to guarantee road safety; they are also referred as *Driver Support Features*. The last three include features not requiring the control of the driver, and they are known as *Automated Driving Features*. More deeply:

- *Level 0*: the automation features are confined to warnings, mainly sound or dashboard signals, and momentary assistance. Examples: Automatic Emergency Braking, Blind Spot warning, Lane departure warning
- *Level 1* ("hands on"): the automation features share the control of the vehicle with the driver for the control of the steering OR brake/acceleration support. Examples: Lane Centering OR Adaptive Cruise Control, Parking Assistance where steering is machine-controlled, while speed is driver-controlled. "Hands on" means that the driver must be always ready to retake the full control. This was the autonomous level of Tesla Autopilot since 2014
- *Level 2* ("hands off"): the automation features support the driver in steering AND brake/acceleration. Examples: Lane Centering AND Adaptive Cruise Control. "Hands off" does not mean that the driver can literally detach his hands from the steering wheel, on the contrary it is steel mandatory, but during the automatic manoeuvre, the driver does not make any actions and he just supervises
- *Level 3* ("eyes off"): the features can drive the car only under well specified conditions, and the driver can safely turn away his attention. However, the driver

must be ready to retake the control within a certain period specified by the car maker since there are still problems in terms of liability. In practice, the autonomous features can be assimilated to a co-driver. Examples: Traffic Jam Chauffeur

- *Level 4* ("mind off"): similarly to the previous level, self-driving intervenes in limited spatial areas or under well specified circumstances, but in addition the driver's attention is not required for safety. Outside those conditions, the vehicle must stop safely. Pedals and steering wheel may not be installed. Examples: robotic taxi or robotic delivery only covering a selected region at a specific time
- *Level 5* ("steering wheel optional"): it is the last step for self-driving vehicles under all conditions [11] [12]

The maximum findable level now on the market is the third.

A variety of sensors are needed for this transition to autonomous world, such as radar, LIDAR (Light Detection And Ranging), sonar, GPS, odometry and inertial measurement units to perceive the surrounding environment [12]. Advanced control systems, including sensor fusion, read sensory information to compute optimized navigation paths, as well as to detect obstacles and relevant signage. The V2X communication is also becoming more and more important for sharing road information between vehicles and enclosing them in safe infrastructural boundaries; governments are supporting the creation of a *Dedicated Short-Range Communication*, and 5G will play a key role in this direction. The result of this interaction will be the "sharing driver" concept for the first commercially available vehicles that will be autonomous just under a certain speed and in specified circumstances. In this contest, the mechanical aspects of the vehicle are losing their primary role, gradually overwhelmed by control algorithms. In fact, many technology companies, such as Google, Apple, Nvidia, are entering in the autonomous automotive world with their own projects [10].

This technology will be also a help for the electric revolution in the automotive world because vehicles would efficiently manage their energy level [10] [12].

In general, an autonomous vehicle can be described by three main interconnected components families, following a "sense-plan-act" design [8]:

- *Sensors*, as described above

- *Logic Process Unit*, which includes software, decision making, checking functionality, user interface
- *Mechanical Control Systems*, consisting of mechanical servo motors and relays, driving wheel control, brake control, throttle control, etc.

Autonomous vehicles also follow a *Layered Modular Architecture* as digital technologies (Figure 3) [13].

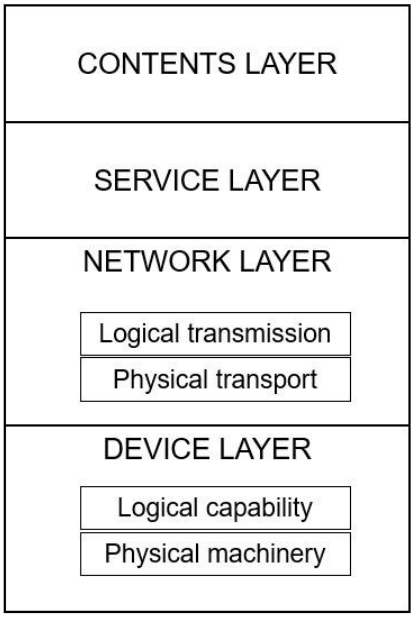


Figure 3. Layered Modular Architecture for digital technologies [13]

- *Device layer*: the *physical machinery* refers to the actual vehicle itself (e.g. chassis, body, powertrain), accompanied by a *logical capability layer* referring to operating systems that make it autonomous. The latter connects it with the other layers, while the former will be revolutionized, in particular concerning the interiors since the vehicle will become a "moving room"
- *Network layer*: the *physical transport layer* refers to the radars, sensors and cables which create the communication skeleton of the autonomous vehicle. Next to that, *logical transmission* contains communication protocols and network standards to communicate the digital information with other networks and platforms or between layers

- *Service layer*: it runs the applications and their functionalities that "serves" the autonomous vehicle, and its owners, as they extract, create, and store content with regards to the driving history, traffic congestion, roads, or parking abilities
- *Contents layer*: it contains the sound and images to improve driving and understanding capabilities [12]

Issues about safety, from external attacks and reliability of the components, and accidents liability are the now the most important obstacles for the spreading of this technology, as well as the revolution of the architectures and cities.

1.4 Autonomous driving: remote control

"A *remote-control vehicle* is defined as any vehicle that is teleoperated by a means that does not restrict its motion with an origin external to the device. This is often a radio control device, cable between control and vehicle, or an infrared controller" [14].

Actually, this kind of vehicles is not used for road transport, but they have many application fields especially in hazardous environments. Space probes, submarines, army robot for buried bombs, automated guided vehicles, and toys are the most common. Many of this examples have the traction and trajectory remotely controlled by joysticks.

Remote-control vehicles can be seen as a grey area of the autonomous world because a person still controlling the vehicle is needed. However, since the passage to level 5 autonomous vehicles will require other 10 or 20 years of research, many companies and universities promote teleoperations technologies for remotely control autonomous vehicles if necessary. In this way, a human operator could monitor a fleet of autonomous vehicles intervening when any of those cars cannot move safely. For example, Google's Waymo tested fully driverless cars in California, provided the presence of remote operators [15].

Cellular network is currently the most used communication technology for remote control. 4G networks are too weak for this purpose because they need vehicles running at a much slower speed to exchange safety information. In fact, network requirements for remote operation include broad coverage, high data throughput, and low latency. Those requirements could enable continuous video streaming and the possibility to send commands between a remote operations center and a vehicle (Figure 4). In this contest, the future 5G will bring a number of benefits to remote control systems, including core network slicing

that will enable priority service provisioning, and radio access to bring ultra-low latency, with a round trip time below 4 ms, and beamforming for high throughput and capacity. Some delays are still present in this possible configuration of radio control, as the video encoding and decoding, and the servo-driven mechanics [16].

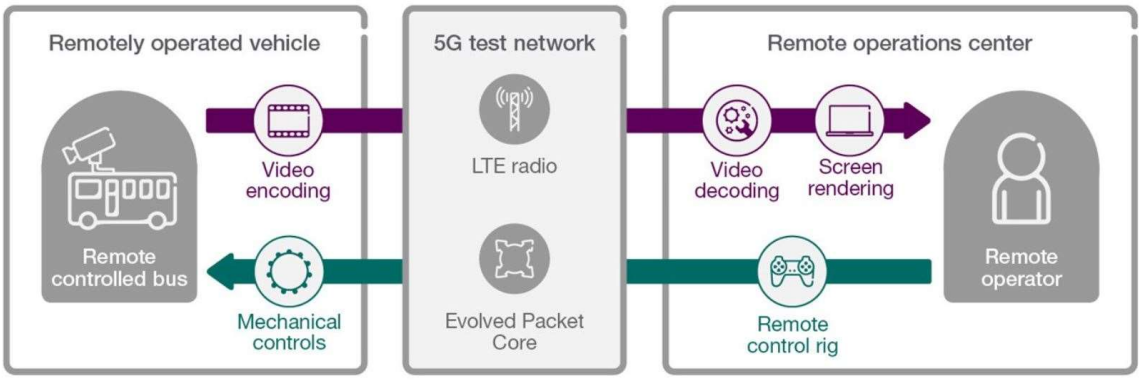


Figure 4. Possible 5G remote control of public transport [17]

In general, communication over a specific network is the most studied methodology to remote control a vehicle. A patent from Toyota Motor Engineering and Manufacturing North America Inc. describes a method to remotely drive a driverless car when it encounters an unexpected driving environment for the autonomous mode, as for example a road construction. The information captured by the sensors, especially cameras and LIDAR, is sent to a *remote server* in a communication network. Additionally, the sent data can be optimized for conserving bandwidth just selecting their fundamental subset. A *remote operator* is then contacted with a request to take control of the vehicle; he receives those data, and he can manually remotely drive in real-time the vehicle until the end of the unexpected environment, or he can just send the commands that the autonomous vehicle should follow to exit from the unwanted situation. Of course, the autonomous vehicle needs enough memory and processing capability to store and execute the commands from the operator [17].

An older patent from Caterpillar Inc. shows the potentiality of using radio communication to interrupt the autonomous operation of a vehicle. The considered autonomous vehicle includes a navigator, a machine control module, an engine control module, a transmission control module, and other systems to permit autonomous operation. In *autonomous mode*, the navigator delivers a speed command and a steering angle command for the vehicle. In

tele-operation mode, a tele-panel communicates the speed and steering angle commands to the navigator through a *radio link*. The navigator then provides these commands to the machine control module to operate the vehicle. Remote control is initiated by establishing radio communication with the vehicle navigator using the tele-panel (Figure 5). The exclusive coupling between the vehicle and the tele-panel is guaranteed by their identifiers that are transmitted by the tele-panel before the tele-driving begins. If the communication is interrupted during this mode, the vehicle enters in a locked state and it halts [18].

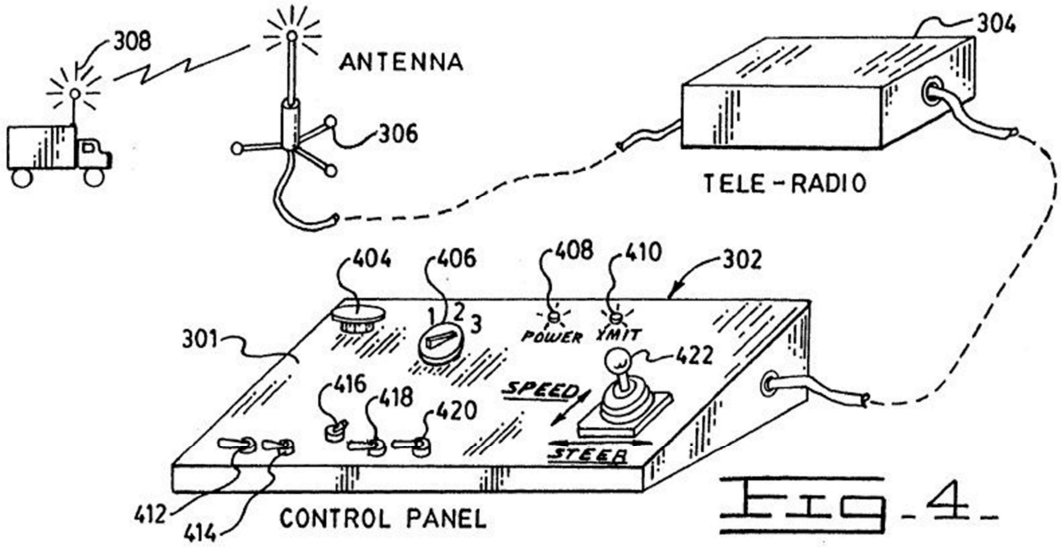


Figure 5. Tele-operation mode of an autonomous vehicle [18]

This last possible remote control of a driverless vehicle is very similar to the discussion in the next chapters about the Remote Emergency System (RES).

1.4 Formula Student Driverless

Formula Student is an educational engineering competition among worldwide universities with the purpose to introduce young engineers into the demanding environment of motorsport competition in terms of design, manufacturing, and financing. The challenge is to design and construct a single-seater formula style race car (Figure 6) that will be judged in terms of engineering design and costs during static events, and in terms of performances during the dynamic events [19].

Figure 6. PoliTo SC19

The main characteristic of this competition is its attention to the future; in fact, there is the possibility to compete either with an internal combustion engine vehicle or with an electric vehicle. Moreover, starting from 2017 in Germany, the Driverless competition has begun with many students involved in the study and design of the new autonomous frontiers of the automotive industry.

The design of the driverless vehicle must follow the rules fixed in the annual rulebook by the organization. Those rules mainly supervise the design of the steering actuator, the Emergency Braking System (EBS), the RES, and all the sensors and control algorithms needed to move the car autonomously. Of course, the whole vehicle must follow the traditional rules also for the mechanical parts and for the traction systems.

1.5 Methodology

This thesis work followed a step-by-step methodology. The study and the mastery of the theoretical basis and of the used software platforms were the first step.

In the first part concerning the *RES*, the thesis is also practice-oriented giving the priority to physical testing of the supplied components in order to understand their response by reverse engineering, and to verify the real functioning of the understudy subsystem.

In the second part regarding the *Autonomous System state machine*, the design of the virtual models starts from basic schemes to verify the acquired knowledge and the initial

behaviours. Solidified those aspects, further progresses focus on giving stronger physical meanings and interacting with the other subsystem of the vehicle.

1.6 Objectives of the thesis

The following thesis has mainly two objectives: to study the behaviour of the supplied *RES* and to integrate in the EBS test bench; then, to design the *Autonomous System state machine* that will interface on one hand the sensors and the *RES*, and on the other hand the steering actuator and the EBS. Both those last subsystems were designed and tuned simultaneously with the other members of the team. The starting vehicle is *PoliTo SC19* (Figure 6), which competed in the Electric Vehicle (EV) class in 2019 and now the Driverless division of the PoliTo Squadra Corse is updating for the Driverless Vehicle (DV) competition.

Chapter 2 – Theoretical bases

2.1 Finite-state machines

As written in the book "Switching and Finite Automata Theory", "a *finite-state machine* (or finite automaton) is an abstract model describing the synchronous sequential machine and its spatial counterpart, the *iterative network*" [20]. A *sequential machine* takes in one set of inputs each clock cycle to produce one set of outputs [21]. Thus, if at a time instant t, an input signal x(t) was to be applied to a machine M, then its response z(t) would depend on x(t) as well as on the past input signals to M. Moreover, since a given machine M might have an infinite basket of possible histories, it would need an infinite memory for storing them. Implementing machines with infinite storage capabilities is of course impossible, so past histories of these machines can affect their future behaviour in only a *finite* number of ways giving to the machine only a finite number of decisional paths [20]. An easy scheme to understand the functioning of a *state machine* is reported in Figure 7 [23].

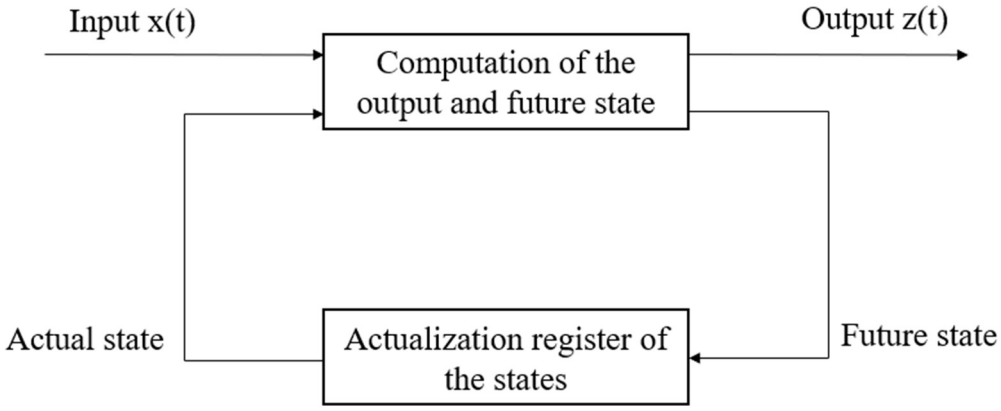


Figure 7. Finite state machine [23]

In details, from the book "Theory of Computing: a gentle introduction", in mathematical terms a *deterministic finite automaton* (DFA) is described by a quintuple $A = \{Q, \Sigma, \delta, s, F\}$ where:

- Q is a *finite set of states*
- Σ is a *finite input alphabet*

- δ is a *transition function* from $Q \times \Sigma$ to Q
- $s \in Q$ is the *initial state* of the automaton
- $F \subseteq Q$ is the set of *favourable states* [22]

The term *deterministic* is used to point out that every action of the automaton is fully determined by its current state and the input.

This book provides also a simple image to visualize a DFA: it can be seen as a movable *reading head* that can read an *input tape* made by a finite number of cells in which there is an input character of the *alphabet* Σ (Figure 8). Initially, the reading head is at the leftmost cell of the input tape. The main part of the automaton is its *finite control device*: at the initial time it is in the state s and at any time it is in a state $q \in Q$. At regular time intervals, the moving head reads $a \in \Sigma$ of the next cell of the input tape and the DFA can pass from the state q to $q' = \delta(q, a)$. Thus, the new state is completely determined by the input and the original internal state. At the end, the moving head reaches the end of the input word; at this moment, if the machine is in a favourable state $q \in F$, the input word is classified as accepted by the machine. Otherwise, the classification is not accepted. The set of all the accepted words by the DFA is called *language* accepted by the automaton, denoted as L(A) [22].

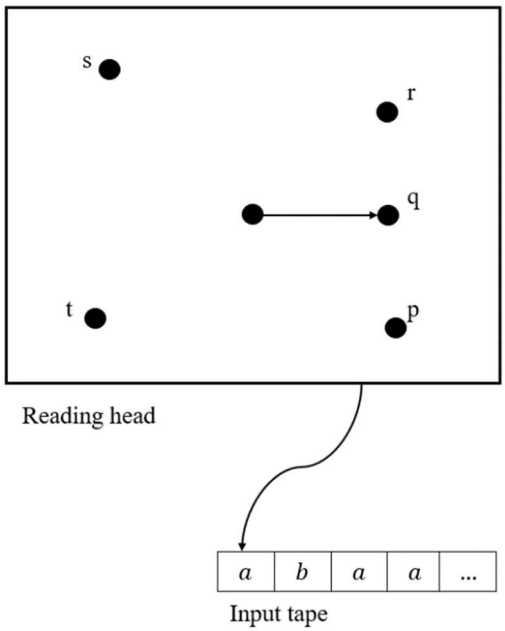


Figure 8. Deterministic Finite Automaton example [22]

The typical representation of a state machine is through a directed graph commonly called S*tate Transition Diagram* (STD) (Figure 9). The *nodes* represent the states, while the *arrows* represent the transitions, and they are labelled with the required input coming from the alphabet of the machine for that transition [22] [23]. The initial state can be highlighted with different methods, in this case the symbol > has been chosen.

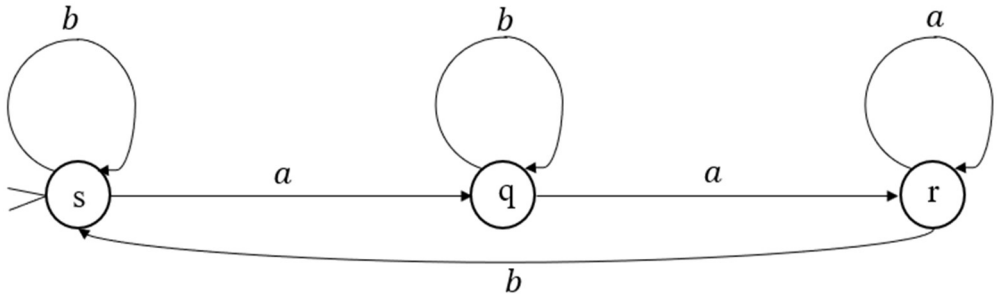


Figure 9. State Transition Diagram [22]

The book "Theory of Computing: a gentle introduction" introduces also the *nondeterministic finite automaton* (NFA). It is a quintuple $A = \{Q, \Sigma, \Delta, s, F\}$ where:

- Q is a *finite set of states*
- Σ is a *finite input alphabet*
- $s \in Q$ is the *initial state* of the automaton
- $F \subseteq Q$ is the set of *favourable states*
- $\Delta \subseteq Q \times (\Sigma \times \{e\}) \times Q$ is the *transition relation* [22]

Every triple (q, a, p) in Δ is represented by an arrow connecting the states q and p and docketed "a" in the state diagram of the automaton A. Two important differences from the deterministic vision can be explained thanks to the state diagram: an arrow can be labelled by the empty string "e" and some states can have no coming out arrows labelled with some symbols $a \in \Sigma$. The empty string means that the NFA can jump from the state q to the state p whenever it wants. In this way, *trap states* can be avoided, which are a typical case of the deterministic machines that remain stuck in a state that can accept all the alphabet returning the current state [22].

It can be demonstrated that for each *nondeterministic finite automaton* A, there is a *deterministic* one A' *equivalent* to A. *Equivalent automaton* means that it can accept the

same language. Many conversion algorithms exist and the decision between deterministic and nondeterministic is made specifically for the case of study since there are no computational advantages [22].

Another important classification is between *Mealy machines* and *Moore machines*.

A *Mealy machine* is a deterministic finite-state machine whose output depends on its actual state and the input [23]. From the mathematical point of view, a *Mealy machine* is a 6-tuple $M = \{Q, s, \Sigma, \Lambda, T, G\}$ where:

- Q is the *finite set of states*
- s is the *initial state*
- Σ is the *input alphabet*
- Λ is the *output alphabet*
- T is the *transition function* $T: Q \times \Sigma \rightarrow$ Q, computing the future state from the pairs of a state and an input symbol
- G is the *output function* $G: Q \times \Sigma \rightarrow \Lambda$, computing the output symbol from the pairs of a state and an input symbol [24]

The typical scheme of a *Mealy machine* is the one in Figure 7, so, in practice, it is the most general case of state machine.

Moore machine, instead, is a deterministic finite-state machine whose output depends just on its actual state (Figure 10) [23]. In computation theory, it is defined as a 6-tuple $M = \{Q, s, \Sigma, \Lambda, T, G\}$, where G is the only difference with the previous definition because it is an output function $G: Q \rightarrow \Lambda$, correlating each state to the output alphabet [25].

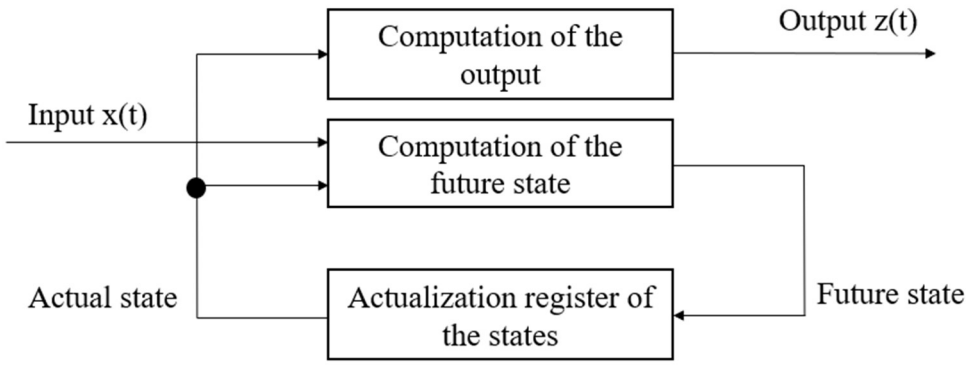


Figure 10. Moore machine [23]

The main advantages of *Mealy machines* are the reduced number of state and the faster response to the input. Nevertheless, *Moore machines* have a better control on the evolution of the machine, in fact output change is always one cycle later with respect to state change, while in *Mealy* ones, input change can cause output change as soon as the transition happens. Consequentially, they result to be easier to test and tune [23] [24] [25].

2.2 MATLAB Simulink Stateflow

Stateflow is a MATLAB Simulink toolbox adapt for modelling and simulating, through a C compiler, state machines and flow charts (Figure 11). It is also possible to analyse and debug the modelled machine thanks to animations and integrity control systems [26].

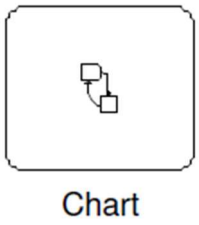


Figure 11. State machine block in Simulink [26]

Once inserted the block in the Simulink environment, it is possible to enter in the development environment of the state machine itself by double-clicking on the block. The design of the machine is based on sketching its *state transition diagram*.

A *finite state machine* is a representation of an *event-driven (reactive) system*. In an *event-driven system*, the system responds to an event by making a transition from one state to another. This transition occurs if the *condition* defining the change is true [26]. Two kinds of states can be modelled in Stateflow:

- *Exclusive states (OR states)*, following the defined hierarchy, those states can work singularly and are characterized by a continuous line (Figure 12)

- *Parallel states (AND states)*, characterized by a dotted line and by a number defining the functioning sequence (Figure 13), they can work both singularly and simultaneously.

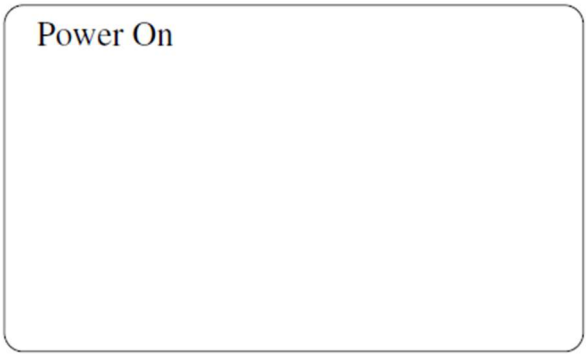


Figure 12. OR state [26]

Figure 13. AND state [26]

Sub-states and sub-machines can be written inside a state.

Some *actions* can be also programmed inside the state mainly linked to the output of the machine. The syntax is "*action_type: action;*". The main action types are:

- *Entry* (abbreviation "en"), the action is executed as soon as the state becomes active
- *Exit* (abbreviation "ex"), the action is executed just before a transition out of the state
- *During* (abbreviation "du"), the action is executed when the state is active, and a specific event occurs
- *Bind*, an event or data object is linked to the state so that only that state and its children can broadcast the event or change the data value

- *On event/message_name*, the event is executed when the state is active, and it receives a broadcast of the specified event or message in the command [26]

Transitions are modelled by arrows exiting from one state and entering in the next one and above it the *actions* and *conditions* of the transition are reported following the syntax: *event_trigger[condition]{condition_action}/transition_action* (Figure 14).

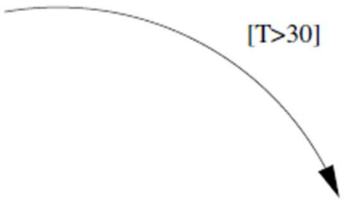


Figure 14. Transition with condition [26]

The *event_trigger* is that event that initializes the transition; it can be also absent and in this case every event can start the transition, or there can be multiple events connected by the logical link OR (|) [26].

The *condition* is a Boolean operation which determines if the transition can happen or not [26].

The *condition_action* is an operation that is executed before reaching the next state if the *condition* is true, even if at the end the state is not reached [26].

The *transition_action*, instead, is an operation executed only if the next state can be reached. Otherwise it is ignored [26].

A particular kind of transition is the *default transition* (Figure 15), really important in an exclusive system since it highlights the starting condition [26].

Figure 15. Default transition [26]

Junctions can be used to eliminate useless states, joining transitions, and exiting new ones with a specific hierarchy (Figure 16) [26].

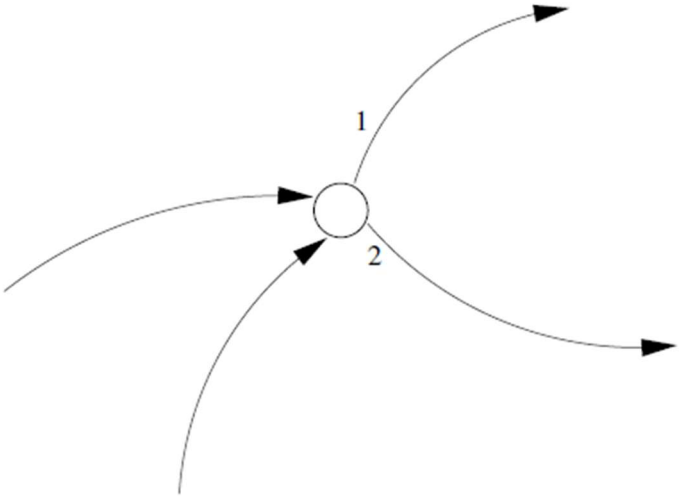


Figure 16. Junction [26]

History junctions exist, too; they are indicated with an encircled "H" inside the state and allow the machine to remember the last sub-state before the transition out of the state. In this way, in case of return in that state, the machine will start from last executed sub-state instead of the default one [26].

Finally, actions and transition conditions can be also linked to *temporal logical operators*. The syntax is *operator(n, E)*. The operators are:

- *After*, which returns "true" if the event E happens at least n times after the state is activated

- *Before*, which returns “true” if the event E happens less than n times after the state is activated
- *At*, which returns “true” if the event E happens at the n-th time after the state is activated, the counter is reset every time the state is activated
- *Every*, which returns “true” every n-th activation of the event E after the state is activated, the counter is reset every time the state is activated
- *TemporalCount(E)*, which increases of 1 the counter and returns the value of the event E every activation of E in the state, the counter is reset every time the state is activated also in this case [26]

Temporal logical operators with absolute time can be used just substituting the event E with the chosen unit of measure, usually seconds (sec) or milliseconds (msec).

The input and output data must be defined in the section *Symbols pane*, based on the symbols used to indicate the events and the conditions. Exiting from the chart, in the Simulink environment, the input and output will become ports that can be interfaced with the usual Simulink tools.

2.3 dSpace Scalexio

dSpace Scalexio (Figure 17) is a family of modular real-time systems for *hardware in the loop (HIL)* and *rapid control prototyping (RCP)* applications. HIL is a testing technique to verify the behaviour of the control models and of the components of a subsystem during the prototyping phase just connecting the electronic control unit to sensors and actuators representing the undertesting subsystem on a bench [28]. RCP is another method to test and iterate the control strategies automatically importing the mathematical models, for example from MATLAB Simulink, to a real-time real I/O machine [29].

Figure 17. dSpace Scalexio family [27]

Main benefits of *dSpace Scalexio* are:

- Scalability to any required processing and I/O requirements
- High-performance processor for fast computation of large and complex model
- Comprehensive, precise, and fast I/O capabilities based on FPGA (Field Programmable Gate Array) technology
- IOCNET (I/O Carrier Network technology optimized for demanding real-time requirements in terms of latencies and bandwidth) real-time backbone with low jitter and high bandwidth for best-in-class closed-loop performance
- Use of several third-party simulation environments via Functional Mock-up Interface (FMI) support [27]

Concerning autonomous vehicles, *dSpace* is suitable for:

- Data replay: “reprocessing of real-world data to validate perception and sensor fusion algorithms”
- Simulation models: “vehicle and traffic simulation featuring ground truth sensors, intelligent driver models, the integration of road networks and environments from HD maps, consistent support of standards, and open interfaces for execution in real time or faster than real time”
- Sensor simulation: “virtual reproduction of the entire sensor transmission channel in order to simulate environments as perceived by cameras, radars, and lidar sensors”

- Sensor fusion test: “validation of perception and sensor fusion algorithms by means of physics-based sensor models, real-time simulation, and synchronous raw sensor data injection into high-performance processing units”
- Radar black-box testing: “over-the-air stimulation of radar sensors with up to four echoes”
- Testing LTE and 5G application in a virtual environment: “open- and closed-loop test setups to provide an end-to-end solution for connected and cooperative autonomous vehicles” [27]

dSpace Scalexio AutoBox (Figure 18) is the compact solution for in-vehicle tests thanks to its improved shock and vibration resistance and an optimized cooling system. Moreover, it has an integrated power supply for 12 V, 24 V, and 48 V electrical systems, 7 I/O slots, and a very precise real-time processing unit [27].

Figure 18. dSpace Scalexio AutoBox [27]

2.4 dSpace MicroAutoBox

dSpace MicroAutoBox (Figure 19) is a compact stand-alone prototyping unit useful also for in-vehicle testing of its major subsystems, as powertrain, chassis, body, ADAS, electric systems, and x-by-wire solutions. It is a real-time system for performing test in *fullpass* and *bypass* scenarios [30].

Figure 19. dSpace MicroAutoBox [30]

Fullpass (Figure 20) means that a new entire ECU with its control logic has to be developed from the start, so the different control strategies can be tested and tuned directly by the prototyping unit at the early stage before producing the physical ECU since modifications after the production would be more expensive, time consuming, and inflexible. In this way, developers can concentrate mainly on the algorithm design without having to worry about memory space and computing power [31].

Figure 20. Fullpass prototyping [31]

In *bypass*-based prototyping, instead, existing ECU software is optimized or partially revised (Figure 21). The original ECU executes all the functions that will remain unchanged, while the new algorithms are calculated in *MicroAutoBox*. The necessary input data and results are exchanged between the *MicroAutoBox* and the original ECU. If the existing ECU already features the I/O data required by the new control strategy, just an appropriate ECU interface with the prototyping unit has to be managed. If the algorithm instead requires additional data, the I/O interfaces of *MicroAutoBox*, to directly connect new sensors or actuators to the vehicle bus [30].

Figure 21. Bypass prototyping [30]

The in-vehicle mounting is ensured by the ISO 16750-3:2007 certification for shock and vibrations. In this way *MicroAutoBox* can be also used directly as vehicle control unit, similarly to a fullpass scenario, as it happens in Polito Squadra Corse SC19. Moreover, there are *MicroAutoBox* variants with interfaces for all major automotive bus systems: CAN, CAN FD, LIN, K/L-Line, FlexRay, and Ethernet. In particular, thanks to the Ethernet port, a PC can be directly hot-plugged for application download, model parametrization, and data analysis. CAN interfaces and topology are easy to configure by means of the dSPACE blocksets RTI CAN and RTI CAN MultiMessage (RTI = Real-Time Interface). The *RTI library* (Figure 22) provides a blockset that lets users implement the functionality and I/O capabilities of *MicroAutoBox* fast and conveniently, directly into controller models created with the development software MATLAB/Simulink/Stateflow [30].

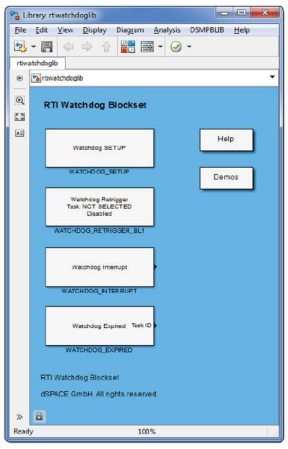


Figure 22. RTI blockset [30]

2.5 Remote Emergency System (RES)

The *Remote Emergency System (RES)* is a standard requirement for Formula Student DV consisting in a *remote control (sender)* and a *vehicle module (receiver)* (Figure 23). The receiver includes a normally-open relay which must be part of the Shutdown Circuit (SDC). The SDC is an electric circuit composed of switches for the activation the main subsystems, particularly the electric traction system, a low voltage battery, and protection components [32].

Figure 23. Remote Emergency System [33]

It has essentially two functions:

- *Stop button*: when pressed, it must trigger the DV Shutdown Circuit (SDC) by a non-programmable logic only manually resettable. Triggering the SDC means performing an emergency braking manoeuvre by the EBS
- *Race-control-to-vehicle communication*: it can send the *Go signal* to the vehicle that replaces the green flags for the race start [32]

The apparatus is provided by Gross-Funk GmbH as a combination of the models GF2000i-codec (*receiver*) and T53R98 (*sender)* based on the application of radio signals to operate a technical device from a separate location as well as to transfer information. The main characteristics are:

- SIL3 (Safety Integrity Level) / EN61508 certification
- 430 ÷ 440 MHz communication band

- 88 mW signal strength
- 12 V or 24 V supply voltage (0.26 A at 12 V) for the *receiver*
- 450 g, 173 mm x 113 mm x 35 mm
- IP20 *receiver* (protection against solids penetration but no from liquids and vapors) / IP65 *sender* (total protection)
- -20 ÷ +60 °C operating temperature, -30 ÷ +80 °C storage temperature [33]

The *receiver* (Figure 24) has a *CANopen* interface with the following properties:

- 1000 kbit/s, 125 kbit/s, 250 kbit/s, and 500 kbit/s in standard configuration
- Cyclic PDOs (Process Data Objects) containing states of switches (*Go signal*) and radio
- Warning 200 ms in advance to shutdown, contained in the cyclic PDO, if a signal loss is detected [33]

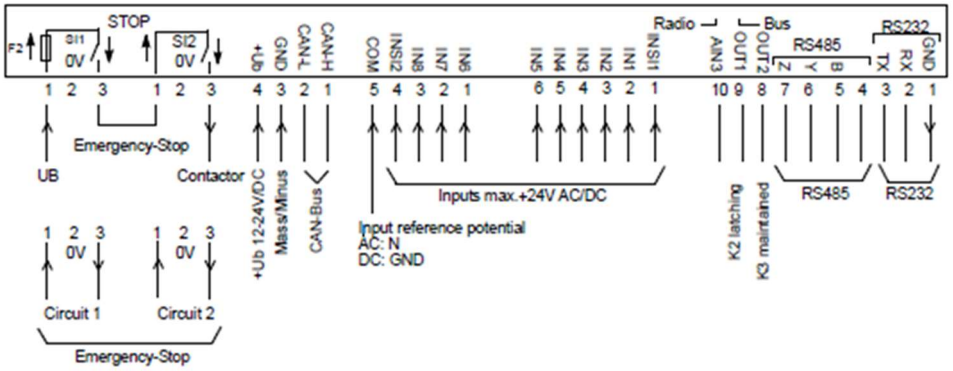


Figure 24. Connections at the RES receiver [33]

The *CANopen* is a communication protocol and a device profile specification for embedded devices in automated systems. Following the OSI (Open Systems Interconnection) model, it implements the network layer and the layers above to the *CAN protocol*, which rules, instead, the physical layer and the data link layer. In particular, the network layer regulates the packet routing in the network. CAN bus can only transmit packets with 11-bit ID, a remote transmission request (RTR) bit and 0 ÷ 8 bytes of data; so, *CANopen* divides the 11-bit ID in 4-bit function code, mapping the used protocols and giving priority to critical functions, and 7-bit *CANopen* node ID [35] [36] [37].

RES receiver communicates with the CAN master device following two protocols: the *NMT* and the *PDO*, standardized CiA 301 standard.

The *NMT protocol* stands for Network Management and it is characterized by a master-slave structure. Through *NMT* services, nodes are initialised, started, monitored, resetted or stopped. A state machine (Figure 25) shows the different modes that the CAN slave, in this case the *RES receiver*, can enter [38].

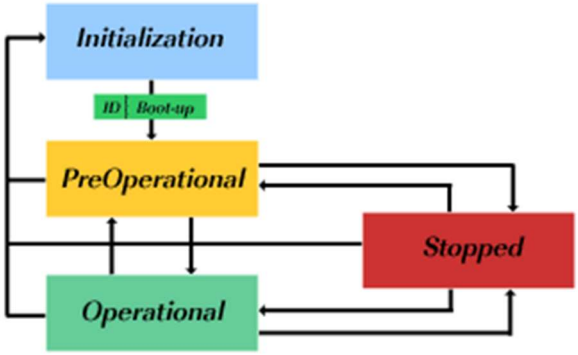


Figure 25. NMT protocol state machine [38]

In the RES case as soon as the slave is booted up, it passes into the *PreOperational state* and singnalizes its initialization to the master. Then the master gives the permission to enter in the *Operational mode* which is the one required for the *PDO* protocol. The *NMT* initialization message is written as CAN-ID = 0x700 + Node-ID + single data byte 0x00. The *NMT* message to pass to the *Operational mode* is of the kind CAN-ID = 0x000, byte 1 = 0x01 (requested state), byte 2 = addressed Node-ID (if the message is for all the nodes byte 2 = 0x00) [33].

Once in the *Operational state*, the *RES receiver* uses *Process Data Object (PDO) protocol* to communicate with the CANopen master because it processes real time data. There are two kinds of PDOs: transmit and receive PDOs (TPDO and RPDO), with RPDO it is possible to send data to the device, and with TPDO it is possible to read data from the device, for example with RPDO it is possible to start a connected device. *PDO* can be also synchronous or asynchronous depending if the transmission happens just after the receiving of a SYNC message from the master [39]. The *RES receiver* transmits cyclic status messages of 8 bytes containing PDOs 2000 – 2007 cyclically every 30 ms. Each PDO has a length of 1 byte, CAN-ID = 0x180 + Node-ID. PDO 2000 contains information about the status of the switch

K2, bit 1, and the button *K3*, bit 2, and, as a consequence, of the *Go signal*. The "Emergency Stop" is signalized by PDO 2000, bit 0, and PDO 2003, bit 7. Moreover, PDO 2006 contains the radio quality, from 0 to 100%, and PDO 2007 summarizes several radio states, for example bit 6 signalizes a pre-alarm radio communication interruption 200 ms in advance to shutdown [33].

The portable *transmitter* is equipped with a LED display indicating the actual operating modes as well as faults in the following manner:

- LED off: transmitter is switched off
- LED flashing permanently: transmitter is switched on
- LED flashing fast: power pack warning, the power level is too low, and the control will turn off after 10 minutes
- LED blinks slowly after switching on the machine: 0-position alert [34]

0-position monitoring is a check against inadvertent or defective starting that can involve both the transmitter and the receiver. For what concerns the *transmitter*, the electronics monitor the operational elements; if they are not in the resting position when the *transmitter* is switched on, it indicates the 0-position alert and avoids transferring the control command. Concerning the *receiver*, it expects a 0-position message after a switching on or a stand-by: if those messages are not received, the *receiver* does not engage any function [34].

Those are all safety specifications particularly important when, depending on local conditions, frequency interruptions can occur, and the machine turn off as "safe mode".

All important and safety functions are controlled by the *dead man's switch* function to avoid any unintentional movement. A *dead man's switch* is a switch that is designed to be activated or deactivated if the human operator becomes incapacitated. Unintentional movements must be avoided in particular if they can start unintended functions.

The radio system has to be safety tested in regular intervals. Two main verifications have to be done:

- The "stop test". It is recommended to be done daily at start of the operations: at the beginning, the transmitter is off, and a function allowing a visible and safe movement is selected; then, the transmitter is switched on, the selected function is pressed and held, and finally, the "stop" button is pressed. The function must stop immediately, and the "stop" button must snap into its place without rebounding

- The “inspection of the dead man’s switch”. At the beginning, the transmitter is off, and a visible and safe function is selected. That function is pressed and held, then the transmitter is switched on, and the machine may not take any movement showing the fast flashing LED (there are also modes with an additional acoustical signal). After that, the function is released, the warn signal stops, and the LED flashes permanently. Pushing again the same function, this must work now as usual
- “0-position of the master switch verification”. Occasionally, the rebound of the master switch to the 0-position must be verified because it can be affected by dirt [34]

Considering the *RES transmitter*, the *Stop button* is a dead man’s switch, and it is used to turn on the device, too, just unlocking it. Inside the *Stop button* there is also the LED for the monitoring of the operating mode. A *turning switch* is also present for selecting the radio channel in case of common initial frequency with another *transmitter*; the *receiver* stops for few seconds during the change.

Finally, the *receiver* is tested against electromagnetic interferences, but malfunctions can still occur due to induction currents, power regulators, frequency converters, high performance radio transmitter, spark gaps, high-voltage generators, power inverter, and similar. Thus, the positioning is constrained by the presence of those elements, and classic suggestion for electromagnetic compatibility should be followed, as short connecting cables and free-wheeling diodes for radio interference suppression [34].

2.6 CAN communication

Control Area Network (CAN) is “an International Standardization Organization (ISO) defined serial communications bus originally developed for the automotive industry to replace the complex wiring harness with a two-wire bus” taking the durability of the system at the same level of the entire vehicle (10÷20 years). The *CAN bus* was developed by BOSCH as a multi-master, message broadcast system that specifies a maximum data rate of 1 megabit per second (bps). Unlike a traditional network, such as USB or Ethernet, *CAN* does not send large blocks of data point-to-point from node A to node B under the supervision of a central bus master. In a *CAN network*, many short messages like temperature or RPM are broadcast to the entire network, which provides for data consistency in every

node of the system. Data consistency refers to the usability of data. The vehicle *CAN network* is made of many different ECUs exchanging information, from the one of the engine to the one controlling the ABS, and so on. Each node can both send and receive messages, but not simultaneously. For non-critical subsystems such as air-conditioning and infotainment, where data transmission speed and reliability are less critical, LIN bus standard is used [40].

Considering the OSI standard, *CAN bus* standardizes just the first two layers: the *Physical layer* and the *Data Link layer*, in turn, divided in *Medium Access Control (MAC)* and *Logical Link Control (LLC)*. Above layers, such the Network layer, are useless because any routing protocol is needed in a closed network where only ECUs are addressed [40] [41].

The *Physical layer* (Figure 26) is characterized by a passive bus topology made by copper twisted pair cables with nominal impedance of 120 Ω. This bus uses differential wired-AND signals; differential means that 2 complementary signals are sent to eliminate noise by subtraction of the signals. Two signals, CAN high (CAN_H) and CAN low (CAN_L) are either driven to a *dominant state* with CAN_H > CAN_L, or not driven and pulled by passive resistors to a *recessive state* with CAN_H ≤ CAN_L. A 0 data bit encodes a *dominant state*, while a 1 data bit encodes a *recessive state*, supporting a wired-AND convention, which gives nodes with lower ID numbers priority on the bus [41].

ISO 11898-2, also called *high-speed CAN* (bit speeds up to 1 Mbit/s on CAN, 5 Mbit/s on CAN-FD), uses a linear bus terminated at each end with 120 Ω resistors. For *high-speed CAN*, any device transmitting a dominant (0) means CAN_H wire towards 5 V and the CAN_L wire towards 0 V, while if no device is transmitting a dominant, the terminating resistors passively return the two wires to the recessive (1) state with a nominal differential voltage of 0 V (receivers consider any differential voltage of less than 0.5 V to be recessive for tolerance). This is the standard for automotive application [41].

ISO 11898-3, also known as *low-speed* or *fault-tolerant CAN* (up to 125 kbit/s), uses a linear bus, star bus or multiple star buses connected by a linear bus, and it ends at each node by a fraction of the overall termination resistance. The overall termination resistance should be close to, but not less than, 100 Ω. *Low-speed fault-tolerant CAN* works in the same way of high-speed CAN, but with larger voltage drops. The dominant state is transmitted by driving CAN_H towards the device power supply voltage (5 V or 3.3 V), and CAN_L towards 0 V, while the termination resistors pull the bus to a recessive state with CAN_H at 0 V and CAN_L at 5 V. The receiver should just consider the sign of CAN_H – CAN_L [41].

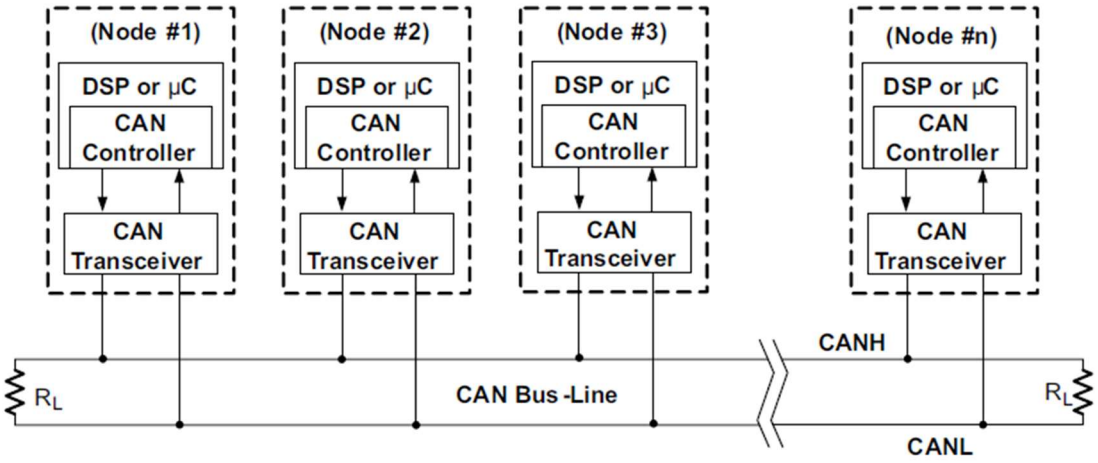


Figure 26. CAN bus physical layer [41]

Moreover, the mechanical aspects of the Physical layer are not formally specified; each car maker or in general each CAN device can have different connectors, often custom [41]. The most common connector is the *9-pin D-sub type* male connector with a pin configuration as shown in the following Figure 27.

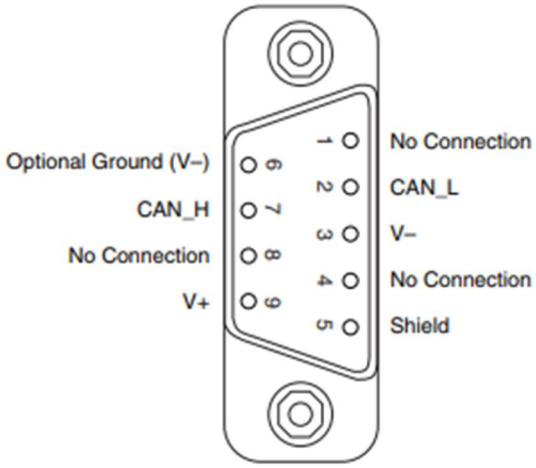


Figure 27. CAN 9-pin D-sub type connector

Passing to the communication engineering field, 4 different kind of packets can be transmitted:

- *Data frame*, useful to transmit information through push mechanism
- *Remote frame*, useful to request data from the other nodes through pull mechanism
- *Error frame*, useful to reset the system informing all the nodes that an error occurred

- *Overload frame*, useful to reserve the bus for very high priority frames deleting what is already in the channel [42]

The *CAN protocol* can follow two different standards for the Data packet structure. The *ISO-11898:2003 Standard*, with the standard 11-bit identifier, provides for data rates from 125 kbps to 1 Mbps (Figure 28). It was later amended with the *"extended" 29-bit identifier* (Figure 29) [40].

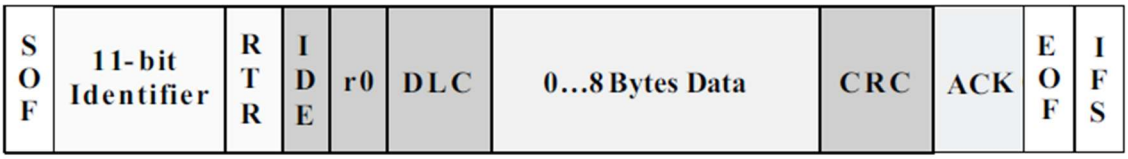


Figure 28. Standard CAN data frame [40]

The meaning of the acronyms in Figure 28 is the following:

- SOF is the *Start Of Frame* and it is a single dominant bit with the aim of reserving the bus for the transmission and synchronization
- *11-bit Identifier*, known also as *Arbitration Field*, it has information about the transmitter, the receivers, and the priority of the message: the lower the binary value, the higher the priority
- RTR is the *Remote Transmission Request* bit and it is an extra-priority bit for data frames with respect to remote frames if the transmitter and receivers are the same
- IDE is the *IDentifier Extension bit* and it is dominant if the frame is in standard format
- r0 is a recessive bit at the transmitter
- DLC is the *Data Length Code* and it is made of 4 bits to indicate the number of bytes (0÷8). It forms the *Control Field* with r0 and IDE
- *Data field*, up to 8 bytes in order to have real time control
- CRC is the *Cyclic Redundancy Check* field with 15 bit of parity check and 1 recessive bit of delimiter
- ACK is the *Acknowledgement field* where the transmitter sends a recessive bit, while the receivers overwrites it with a dominant bit only if the CRC has not any error. A recessive final bit works as delimiter

- EOF is the *End Of Frame* with 7 bits and it must be recessive, and it disables bit stuffing, that is a technique to ensure enough transitions to maintain synchronization inserting a bit of opposite polarity after five consecutive bits of the same polarity
- IFS is the *InterFrame Space* with 7 bits containing the time required by the controller to move a correctly received frame to its proper position in the network [41]

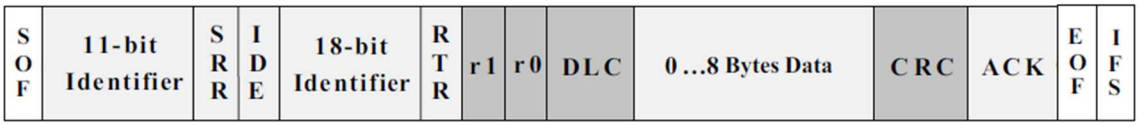


Figure 29. Extended CAN data frame [40]

Figure 29 shows the data frame for the *Extended CAN protocol*. Advantage of this new protocol is the increment of the maximum nodes in the network since the Identifier can assume 2^{29} values. It has the same standard message with these additions:

- SRR is *Substitute Remote Request* which substitutes the RTR in the position of the Standard frame. It is a recessive bit in order to give priority to Standard frame if the 11-bit identifier is the same
- IDE is soon after the SRR and it is a recessive bit for Extended CAN data frame
- r1 is an additional reserve bit that is recessive in case of Extended format [40]

The *Remote frame* is essentially equal to the *Data frame* without the Data field [42].
Bus errors are bits errors, stuff errors, form errors, ACK errors, and CRC errors [42]. The *Error frame* is composed of 3 fields:

- the *Error flag* that is a field with 6 consecutive dominant bits to interrupt and destroy other communications in the bus. All the nodes in the network receive it and respond with 6 recessive bits
- the *Error delimiter* with 8 additional recessive bits when the bus error has been detected
- the *Interframe space* with 3 bits [42]

The *Overload frame* is similar to the Error frame [42].

The *MAC layer* regulates the encapsulation of the data, the access to the common medium, and errors control. The *CAN bus protocol* is a *Carrier-Sense Multiple-Access protocol* with

Collision Detection and Arbitration on Message Priority (CSMA/CD+AMP). CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. CD+AMP means that collisions are resolved through a bit-wise arbitration, based on a pre-programmed priority of each message in the identifier field of a message. The higher priority identifier always wins bus access [40] [42].

The *LLC layer* has the functions of overload notification and acceptance filtering, alongside the multiplexing function [42].

Many implementations of CAN bus-based higher-level protocols exist. *CAN in Automation (CiA)* is the international users and manufacturers organization that develops and supports CAN-based higher-layer protocols and their international standardization, and CANopen can be found among these protocols [40].

Chapter 3 – RES testing and implementation

3.1 Test bench

At beginning of the driverless vehicle assembling, a simple test bench (Figure 30) was prepared in order to activate the supplied RES and to understand its real performances.

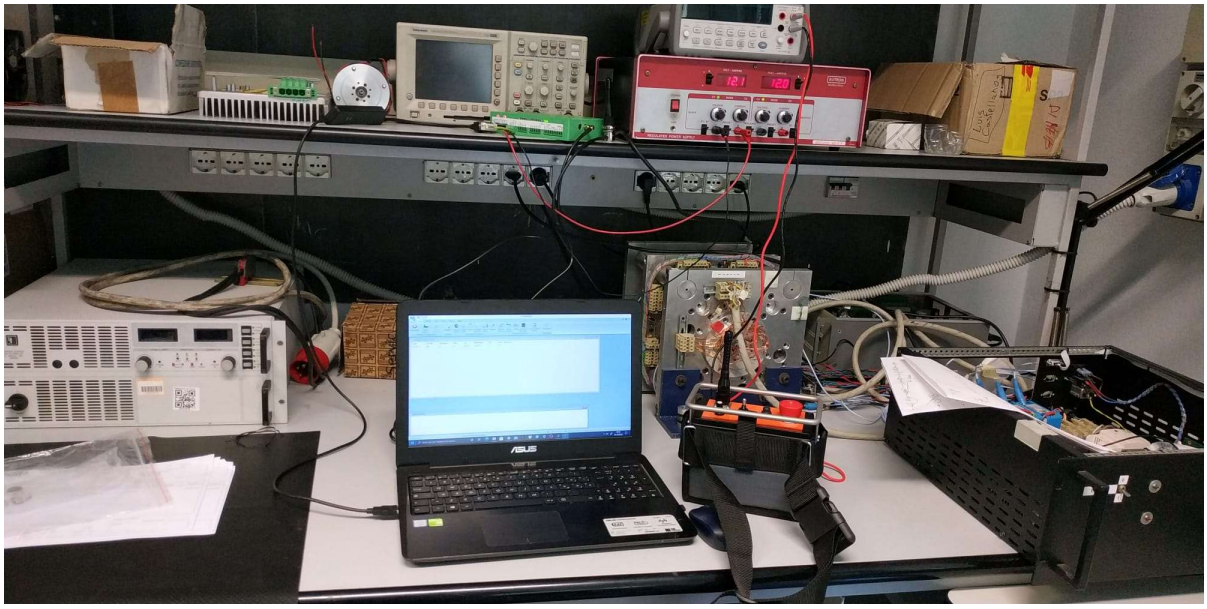


Figure 30. RES test bench

The test bench arrangement is the following: Eutron power supply (12 V), connecting electrical cables, RES receiver, CAN db9 connector, PCAN-usb adapter, and pc. The cables are connected to the power supply through pins and to the RES receiver through a screw terminal block. The DB9 connector is welded to two small cables fixed in the previous terminal block.

The voltage supply is an Eutron D.C. Power Supply Low Power 1-n Output (Figure 31) that allows to impose the voltage handling the knobs and monitor the current flowing. The cables connection is done by pins, while in the receiver the cables are fixed thanks to a screw terminal block (Figure 33). The imposed voltage is 12 V. Above it, there is an Agilent digital multimeter5 (Figure 31) useful to measure electric signals in the output ports of the *RES receiver*. In fact, triggering the toggle switch *K2* or the push switch *K3* translates into output voltage of 12 V respectively from output ports 1 or 2.

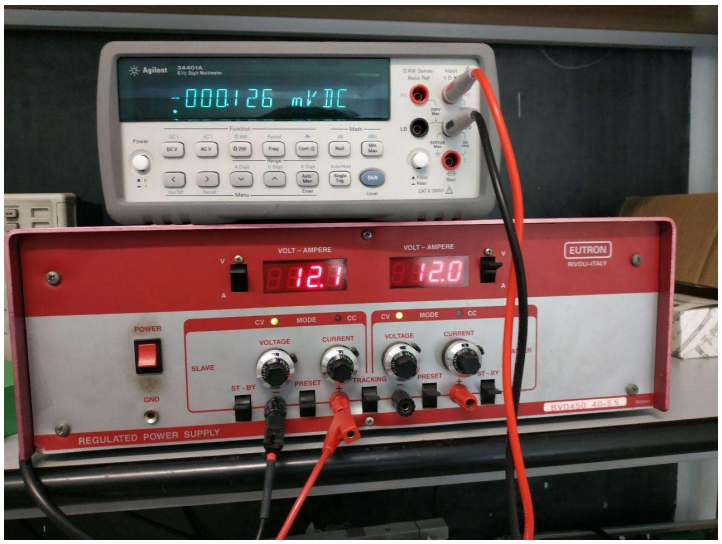


Figure 31. Eutron voltage supply and Agilent multimeter

The *RES receiver* is set to node ID 0x011 as specified in the FSG Competition Handbook [33] with a chosen baud rate of 1 Mbit/s. The setting is done by the DIP switch (Figure 32) on one side of the receiver. Moreover, the CAN connection (Figure 33) has on one side the screw terminal block and on the other two small cables welded to a CAN DB 9 female in the pins 2, CAN low, and 7, CAN high.

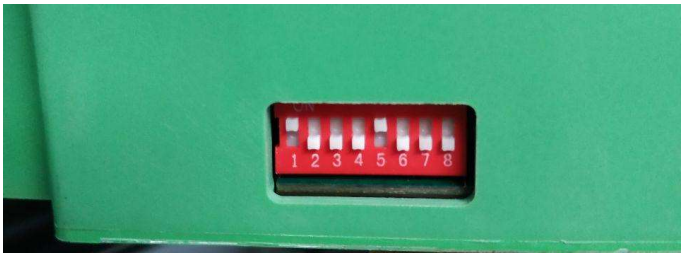


Figure 32. RES DIP switch

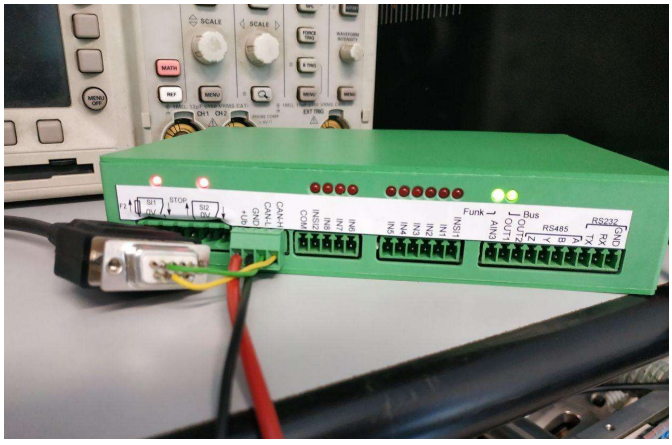


Figure 33. RES receiver, cabling, and CAN db9 connector

A PCAN-usb adapter Kvaser Leaf Light v2 (Figure 34) is directly attached to the CAN DB 9 female on one side and to a PC on the other. It allows to include in the CAN network also the PC with a maximum performance of 8000 messages per second, each time-stamped with 100 microsecond accuracy.

Figure 34. PCAN-usb adapter Kvaser Leaf Light v2

Their own antennas are also mounted on the *transmitter* and *receiver* and the radio frequency is chosen as channel 1 of the *RES*.

The first activity concerned the registration of the CAN messages sent by the *receiver*. Busmaster is the software used to couple the PC with the created CAN network. As soon as the receiver receives voltage, it sends the boot-up *NMT* message with CAN-ID=0x711, as forecasted considering the previous chapter, DLC=1, and Data byte=00, as indicated in the FSG Competition Handbook [33].

Then, the verification of the *PDO* communication followed, but due to the necessity of CAN master, an additional step was needed; thanks to MATLAB Simulink and the Vehicle Network Toolbox, a fictious master was programmed to send a message with ID=0x000, DLC=2, and Data bytes=01 00 (Figure 35). In this way, the PC can send the *NMT* message to pass in the *Operational state* to all the nodes in the CAN network for simplicity. In this case, the *RES receiver* is the only node in the network.

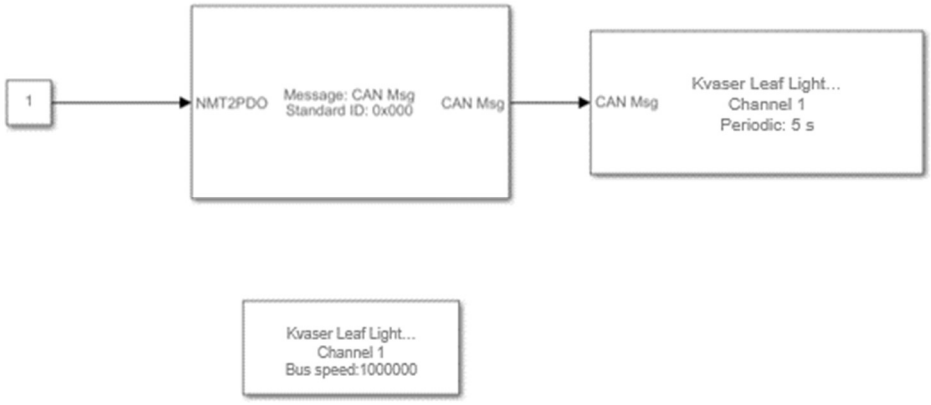


Figure 35. Simulink fictious CAN master

The isolated block is *CAN Configuration* which defines the channel and its baud rate. The channel can be physical, as in this case of study, or virtual thanks to the software itself. Then, the block in between is the *CAN Pack* useful to translate a MATLAB signal into a CAN message. Three main roads can be taken: use of the raw data, manual definition of the message choosing the ID, the DLC, and the related signals, and definition of the message through DBC file. Finally, there is the *CAN Transmitter* that is in charge of the message insertion in the CAN network, once imposed the channel and the typology of transmission.

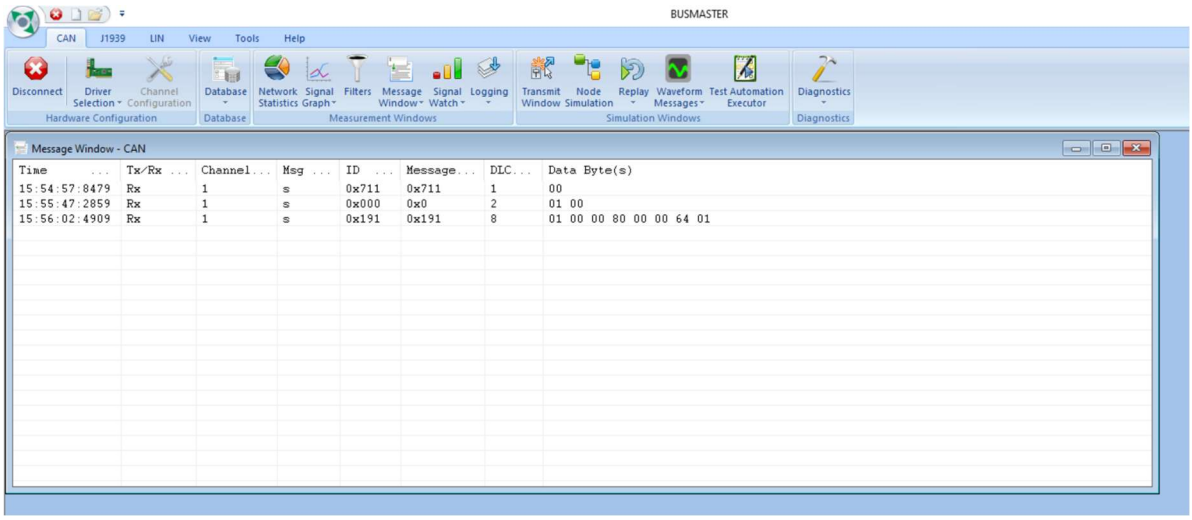


Figure 36. Busmaster interface

Once the *NMT* node control message is sent, the *RES receiver* starts sending cyclic PDOs every 30 ms to update the CAN master about its state. Acting on the *RES transmitter*, we can read 4 different messages thanks to Busmaster (Figure 36). Those messages are reported in the following table:

ID	Message	DLC	Data Byte (Hex)	K2	K3	STOP
0x191	0x191	8	01 00 00 80 00 00 64 01	off	off	off
‘’	‘’	‘’	03 00 00 80 00 00 64 01	on	off	off
‘’	‘’	‘’	05 00 00 80 00 00 64 01	off	on	off
‘’	‘’	‘’	00 00 00 00 00 00 00 00	off	off	on

The respect of the indication in the FSG Competition Handbook about the PDOs is guaranteed. In fact concerning PDO 2000, so the first byte, 03 (hexadecimal) = 0000 0011 (binary), so the bit 1 is equal to 1 when the switch *K2* is in ON position, 05 (hexadecimal) = 0000 0101 (binary), so the bit 2 is 1 when the switch *K3* is pushed. Moreover, in PDO 2003, 80 (hexadecimal) = 1000 0000 (binary) because the last bit shows that the *Stop button* is not pushed. In PDO 2005, 64 (hexadecimal) = 100 (decimal), therefore the radio quality is 100% due to the proximity of the transmitter and receiver on the bench. The last message of zeros is linked to the pushing of the *Stop button* and the consequent opening of the *receiver* internal relay [33].

3.2 DBC file definition and testing

A more realistic and simpler simulation of the *RES* communication can be done using a *DBC file*. A *DBC file* is an “ASCII (American Standard Code for Information Interchange) based translation file used to apply identifying names, scaling, offsets, and defining information, to data transmitted within a CAN frame”. It is essentially a database and the most common way to handle identification and translation of data from the raw CAN message to a meaningful physical value. For any given CAN ID, a DBC file can identify some or all the data within the CAN frame [43].

Based on the registered exchanged packet in the *RES* communication, the writing of a specific *DBC file* was possible. The used freeware was VectorCANdb ++ Editor. This

software offers the possibility to create a CAN database for different standards adapting the template. For sake of simplicity, the basic CAN template was chosen. The organization of the software follows a hierarchical tree (Figure 37) that can be exploited with a top-down or a bottom-up methodology.

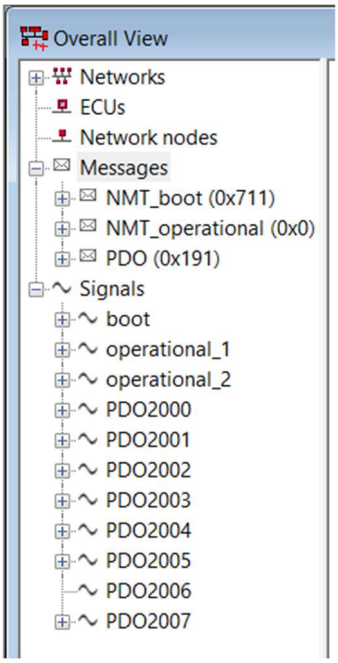


Figure 37. DBC hierarchical tree

In this case of study, the definition of the network was not necessary; so, following a top-down approach, the messages were the first defined objects. As Figure 37 shows, three kinds of messages are stored in the *DBC file* and they are characterized by the ID and the DLC:

- *NMT_boot*: CAN-ID = 0x711 and DLC = 1, it is the NMT boot-up message
- *NMT_operational*: CAN-ID = 0x0 and DLC = 2, it is the NMT message to pass into operational state
- *PDO*: CAN-ID=0x191 and DLC = 8, as the name suggests, it codifies the PDO communication

Then, the signals were defined. Each 8-bit signal is linked to a message. For example, as it is possible to read in Figure 37, 8 PDO signals are written in the *DBC file* because the PDO message contains 8 bytes of data. Signals are characterized by their number of bits, value type, offset, maximum and minimum values. In case of finite number of values, the signal

can be defined by *value tables*. Considering the *RES* communication, the following signals are present in the *DBC file*:

- *boot*: linked to the message NMT_boot, it has 8 bits, and it is defined by a value table in which the value 0x0 is translated as "ON"
- *operational_1*: it is the first byte of the message NMT_operational, and it is defined a value table in which the value 0x1 is translated as "operational_mode"
- *operational_2*: it is the second byte of NMT_operational, and its value table defines 0x0 as "all nodes", since in this case the message would be sent to all the nodes in the network, and 0x11 as "RES_ID", because in this case the message would be sent only to the RES receiver
- *PDO2000*: it is the first byte of the message PDO, and its value table defines 0x0 as "OFF", 0x1 as "ON", 0x3 as "K2_GO", and 0x5 as "K3_GO". The meaning of the values is explained in the table in the previous paragraph
- *PDO2001*: it is the second byte of the message PDO, and its value table defines 0x0 as "standard", since there are no other signals emitted by the receiver in this byte
- *PDO2002*: it is the third byte of the message PDO, and its value table defines 0x0 as "standard", since there are no other signals emitted by the receiver in this byte
- *PDO2003*: it is the fourth byte of the message PDO, and its value table defines 0x0 as "E-STOP" and 0x80 as "ON"
- *PDO2004*: it is the fifth byte of the message PDO, and its value table defines 0x0 as "standard", since there are no other signals emitted by the receiver in this byte
- *PDO2005*: it is the sixth byte of the message PDO, and its value table defines 0x0 as "standard", since there are no other signals emitted by the receiver in this byte
- *PDO2006*: it is the seventh byte of the message PDO, and it is defined with unsigned value type, so all positive numbers, minimum = 0 and maximum = 100, unit = %. This byte describes the radio quality in percentage, as also written previously
- *PDO2007*: it is the last byte of the message PDO, and its value table defines 0x0 as "E-STOP" and 0x1 as "ON"

It is easy to notice that the codification of the signals follows the registered messages thanks to Busmaster. Having the *DBC file* allows to *bypass* the CANopen communication protocol because the messages can be already tracked in the CAN bus lower layers.

Once prepared the *DBC file*, a further step in the simulation was done including the blocks *CAN Receiver* and *CAN Unpack* and defining the CAN messages directly with the *DBC file*. In this way, the Simulink model can also work as receiver (Figure 38). The first block, once defined the channel and the sample time, has as outputs the CAN message and a trigger to a Function-Call subsystem. To translate the CAN message, the Function-Call subsystem must include the block *CAN Unpack* which in practice works symmetrically to the *CAN Pack* with the same possible configurations.

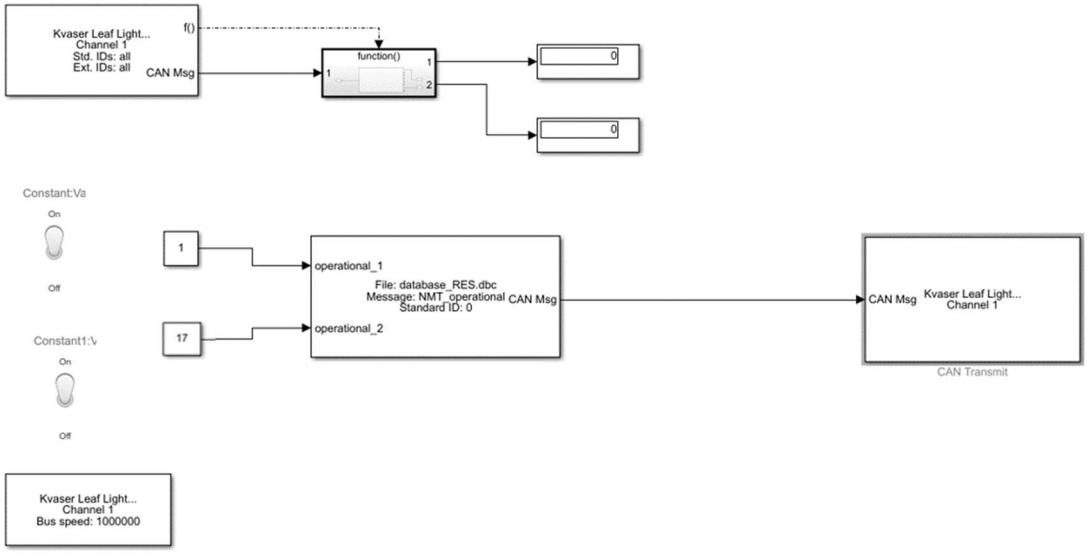


Figure 38. NMT transmission and PDO receiving

In this simulation, two fictious switches are useful to initialize the sending of the NMT_operational message. The block with the constant 17 (decimal) means that the communication is directed only to the *RES receiver*. The time of the simulation is infinite because the information coming in the cyclic PDOs can be read in real time. The most significant PDOs, the 2000 and the 2006, are also displayed to understand the state of the receiver. In the case of the Figure 38, the two zeros displayed means that the *Stop button* is pushed.

Last step in the Simulink environment was the simulation of the entire communication cycle (Figure 39) developing the previous model in Figure 38. As a matter of fact, another *CAN receiver* was implemented in the model to receive the NMT boot-up message and to elaborate it into a signal for the NMT transmission. This elaboration consists of using the signal *boot* exiting from the *CAN unpack* to send the signals *operational_1* and

operational_2. Adding 1 and 17 respectively to the received 0, it is possible to create the two data to translate in a CAN messages with the *CAN Pack* block.

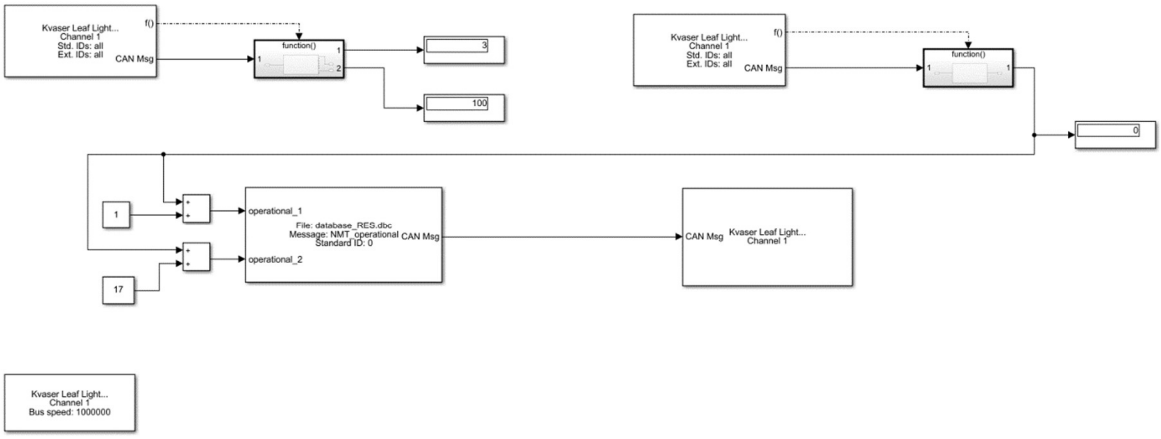


Figure 39. Complete RES communication model

In figure 39, the displayed 3 means that the switch *K2* is in ON position and the radio quality is 100%; in fact the switch was physically triggered, and the radio quality was at the maximum because the receiver and the transmitter were very close during the test.

3.3 Integration in the EBS and testing

Firstly, in order to understand in the proper manner the following discussion, the designed *EBS* of the SC19 must be described both in its fluid-mechanical part and electrical circuit. The design starting point is always the FS Rulebook; it requires only the use of passive systems with mechanical energy storage, and it can be part of the hydraulic brake system [32]. In the SC19 case, the hydraulic braking system is already an emergency braking system since the 90% of the braking pedal travel activates just the regenerative braking for the maximum energy recovery during manoeuvres. After many redesign of a braking pedal actuator, the team shifted to the direct activation of the braking lines in order to fulfil the redundancy requirements. As Figure 40 shows, the *EBS* is fully redundant because it is built on the redundancy of the braking system itself that has two separate hydraulic circuits for the front and rear wheels.

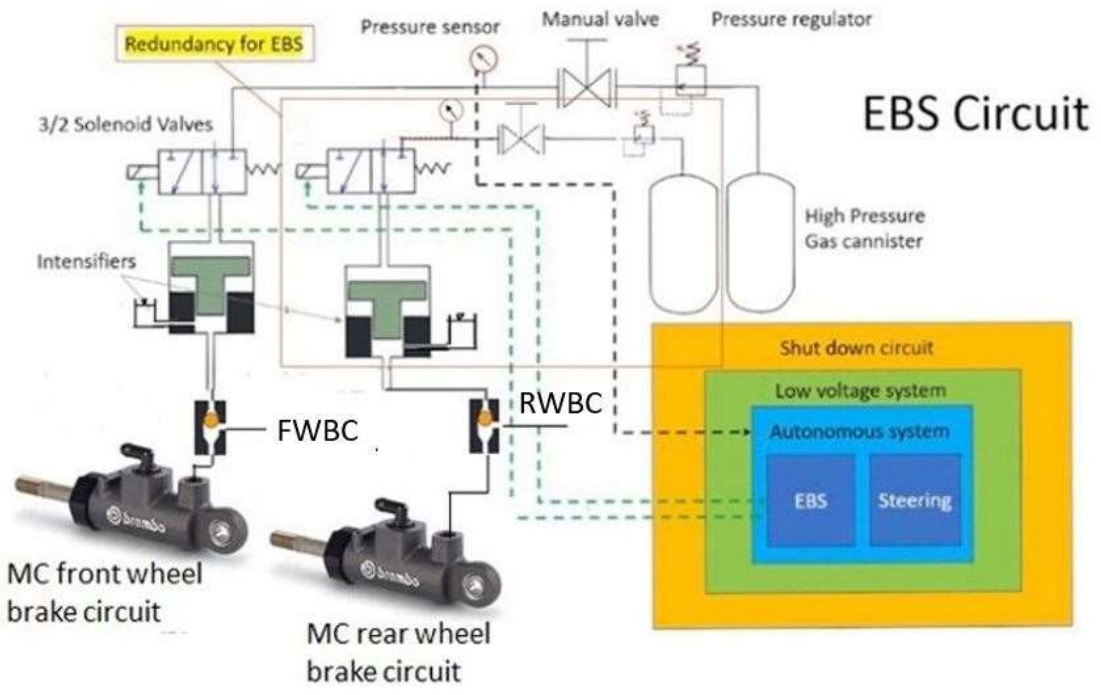


Figure 40. Hydro-pneumatic EBS scheme

The mechanical energy storage is a *high-pressure canister* of 9 cubic inches usually used in paintball and soft-air guns. The air inside is stored at 200 bar, and the first pressure regulation is done directly by the output port of the canister to 58.6 bar. Then, another *pressure regulator* is mounted on the canister to bring the pressure to 10 bar. The pneumatic line starts from this last regulator and finds a *3-ways 2-positions manual valve*; this valve is useful to put the EBS in *armed state* turning the valve in the position connecting the input port from the canister to the rest of the pneumatic line, and it is necessary also to manually discharge the system after an emergency stop connecting the port for the pneumatic line to the external environment. This valve has also an intermediate position that locks both the outputs. The next pneumatic component is a *normally open 3-ways 2-positions solenoid valve*; it has the goal of locking the passage of the compressed air during autonomous or manual driving and opening for *EBS* activation. In fact, the following component is a *hydro-pneumatic intensifier* that converts the pneumatic pressure into hydraulic pressure boosting it for the disc brakes. It works thanks to its geometry; the compressed air moves a cylinder with a bore of 40 mm until closing the connection to the master cylinder oil reservoir, which leads to the increase of the oil pressure thanks to the reduction of the bore to 18 mm of the rod in the oil side of the intensifier. In this way, the 10 bar of pneumatic pressure ideally transforms into

49 bar of oil pressure at the equilibrium ($p_{oil} = p_{air} \frac{d_{air}^2}{d_{oil}^2}$). Before reaching the pistons of the disc brakes, there is a *shuttle valve* useful for the connection to the existing braking lines as output, but also to preserve the functioning of the braking pedal during manual driving.

The *3/2 solenoid valves* are supplied by the *Low Voltage Circuit* of the vehicle. This circuit is powered by a 24 V battery and it includes all the electrical components not involved in the creation and release of traction power. As the FSG Guidebook suggests, a *normally open relay* should be present upstream the two parallel *solenoid valves* [33]. This electrical component works as a switch; if its coil is excited by the *Shutdown Circuit* (SDC), it is closed to let the Low Voltage battery supply the solenoid valves, if the SDC opens in a point, the current doesn't flow anymore in the coil and the relay opens interrupting the voltage supply to the solenoid valves, so activating the *EBS*. The *Shutdown Circuit* (SDC) (Figure 41) is a safety circuit needed to halt the *High Voltage System*, in charge of the traction, in case of any danger directly by the buttons on the dashboard for the driver, internal switches (BOTS, BSPD, …), and the Accumulator Isolation Relays (AIRs), needed to break the electric circuit of the high voltage battery pack in the case of crash, short circuit, overheating, and other emergencies. The *RES receiver normally open relay* is also inserted in the SDC in order to lock the traction system and to initialize an EBS manoeuvre [32].

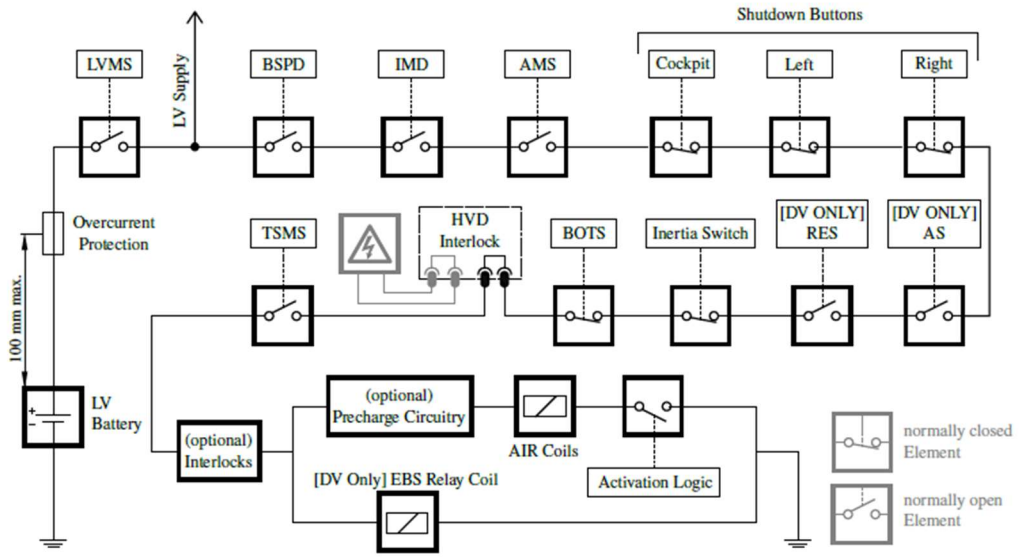


Figure 41. Shoutdown Circuit scheme from FS rulebook [32]

The test bench for the integration of the *EBS* with the *RES* followed the CAD positioning for the hydro-pneumatic components and their supports (Figure 42), while the electrical cabling respected the connections of the two circuits. Nonetheless, the cables length was not the same as it will be in the vehicle, and the power supplies for the two circuits were independent for connections simplicity. So, on one hand there was the circuit of the *relay switch* and the *solenoid valves* and on the other hand there was the connection between the *RES receiver relay* and the *EBS relay coil*. The *RES receiver* was supplied independently to avoid making parallel connections.

Some differences in the configuration of the test with respect to the designed physical implementation were still present. The compressed air was supplied by a compressor directly connected to the tested line to have infinite air charges, and the *intensifier* was locked with a bolt at the output port since the hydraulic lines were not delivered yet. The air pressure was lowered to 5 bar to keep the team safe during the first tests and to avoid damaging the piston inside the intensifier; in fact, if it reached the end stroke, it would hit the bolt with a very high concentrated force even for a steel component. The pressure sensor in Figure 40 was substituted with a *pressure switch* because its purpose is just to inform the central control unit that the *EBS* has pressure in the line (*armed state*) or not. So, a *pressure switch* with a pressure threshold equal to the minimum one needed for a safe *EBS* manoeuvre is enough. Its output is a digital signal for the control unit, and it is captured by an oscilloscope in the test bench. It is pneumatically connected to the circuit with a T joint and its presence is a definitive design choice also for the final mounting.

Before connecting all the hydro-pneumatic components, a simple test was performed to verify the electrical cabling. The air supply was disconnected but the verification of the functionality was confirmed by the internal movement of the solenoid valve when it returns in the normally open position due to current interruption. The movement was shown by a perceived sound. The opening of the supply circuit of the solenoid valve is done by the normally open *relay* which opens when there is no more current in its coil. The coil is supplied by the second circuit where there is the *RES receiver relay* connected in series; when the *Stop button* is pressed, the *RES relay* opens interrupting the current in the EBS relay coil. The connection between the *RES receiver* and the *EBS relay* is shown in Figure 43.

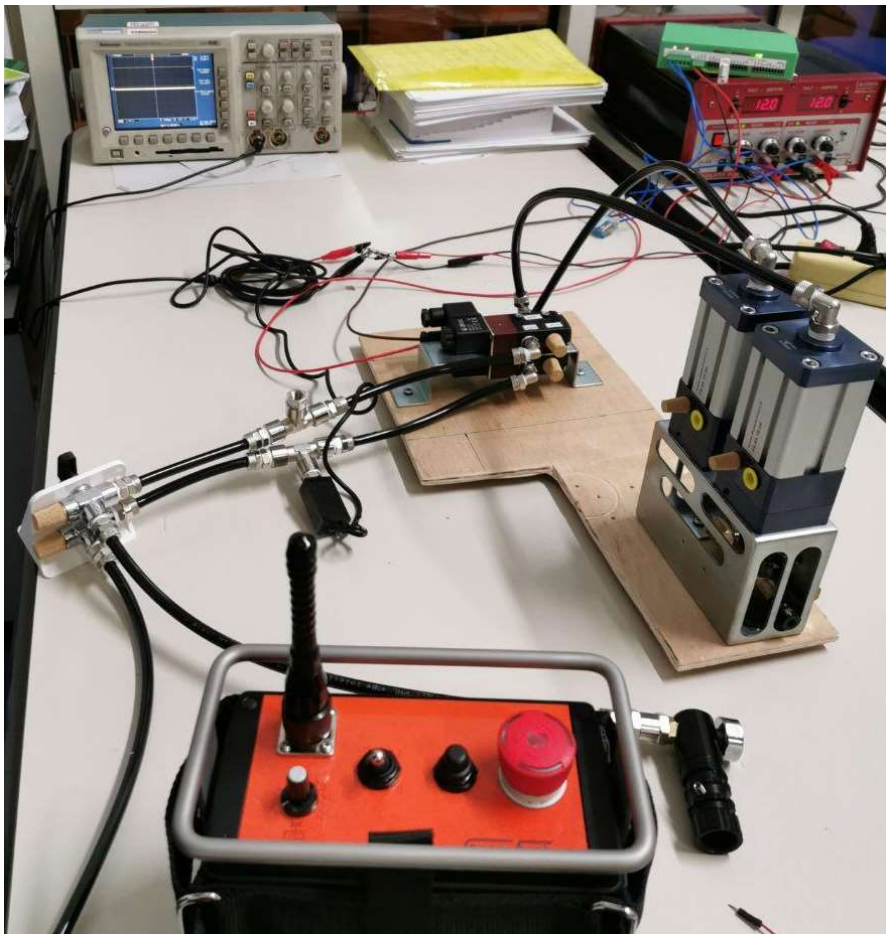


Figure 42. EBS and RES test bench

The EBS and RES test bench has the following arrangement: Eutron power supply (12 V), RES receiver, EBS relay, 3/2 solenoid valves, intensifiers, pressure switch, oscilloscope, manual valves, and a compressor (not in photo). Concerning the electrical cabling, there are two main circuits and other two just for the supply of the pressure switch and of the RES receiver, respectively. The first main circuit starts from the power supply, enters in the RES receiver relay through a screw terminal block, passes into the EBS relay coil, and returns into the power supply for the connection with the ground. The second one starts from the power supply, too, enters in the EBS relay switch, then in the EBS solenoid valve, and returns into the power supply for the ground connection. The compressed air flows from the compressor to the intensifier, passing through a manual valve useful to charge/discharge the circuit, a T-joint for the pressure measure, and the solenoid valve for the activation of the system.

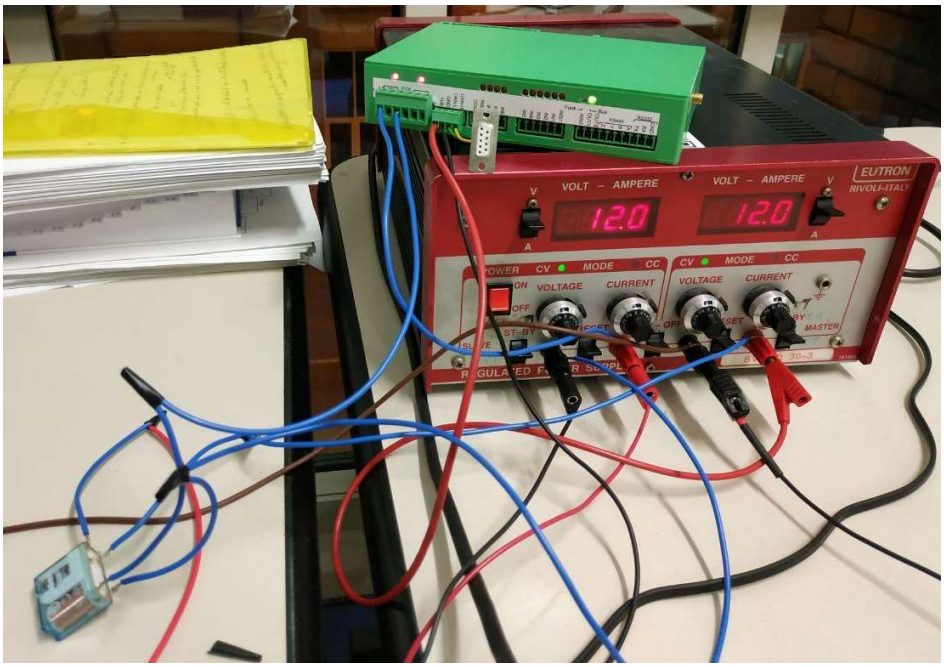


Figure 43. RES and EBS connection through the EBS relay

Then, the main test was performed connecting all the pneumatic pipes as it is possible to see in Figure 42. The first action was opening the connection with the compressor but keeping the manual valve in off-position to build-up pressure in the first part of the air line. Then, the stand-by of the power supplies was released and the pressure switch signalled 0 V since there was no pressure downstream the manual valve, so the EBS was *disarmed*. Some white noise can be also seen from the oscilloscope signal (Figure 44).

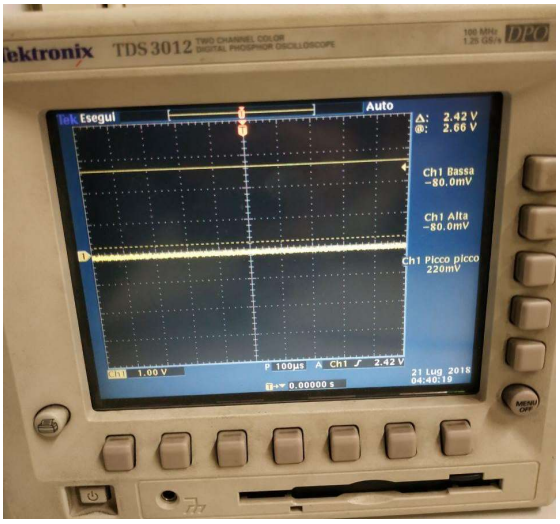


Figure 44. Oscilloscope signal: EBS disarmed (0 V)

Then, the manual valve was turned to connect the input to the EBS line. The pressure build-up helps the commutation of the purchased 3/2 solenoid valve because it is piloted by an input pressure of at least 4 bar. At this moment, the pressure switch was signalling 5 V, so *EBS armed* (Figure 45). Of course, some disturbances of the electrical signals were shown by a non-perfect value of the voltage (5.12 V).

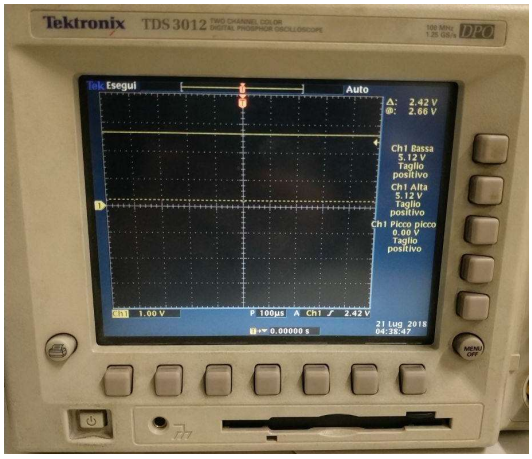


Figure 45. Oscilloscope signal: EBS armed (5 V)

Finally, the *Stop button* was pushed, and simultaneously the solenoid valve returned in the open position giving pressure to the intensifier. The activation was noticed also thanks to a strong sound similar to a gunshot of a compressed-air gun due to the strong compression of the air entrapped in the space between the intensifier rod and the closure bolt. Despite a relatively low output pressure around 24 bar by manual computation, a very high force was produced such to move of some millimetres the bolt out of the output threading.

Releasing the *Stop button*, all the power supplies were repristinated, and the solenoid valve discharged the intensifier thanks to the connection between the intensifier line and the output silencer in the triggered position.

Chapter 4 – Autonomous state machine design

4.1 Formula Student autonomous state machine

The Formula Student rulebook 2020 strictly regulates the design of the *Autonomous state machine* for the driverless vehicle in section DV 2.4. It is mandatory to implement in the Autonomous System (AS) of the vehicle the imposed state machine (Figure 46) without any other states or transitions [32].

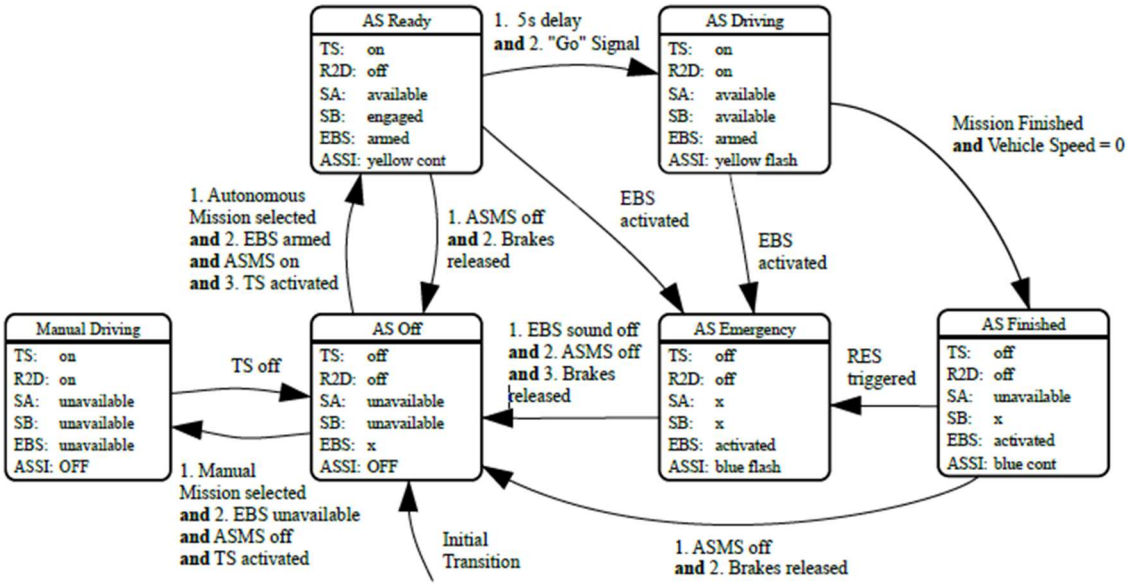


Figure 46. AS state machine [32]

Before entering in the details of the state, a list of the acronyms and the functionality of the named systems is useful:

- TS: Tractive System
- R2D: Ready-to-Drive, it is typical of the electric vehicles and it means that the motor can respond to the position of the accelerator pedal, measured thanks to a position sensor
- SA: Steering Actuator
- SB: Service Brake
- EBS: Emergency Brake System

- ASSI: Autonomous System Status Indicator, it is a LED lamp indicating the actual state of the vehicle. There are three ASSIs on the vehicle; two must be located on each side of the vehicle behind the driver's compartment in a specific region below the main hoop, while the other one must be positioned in the rear, on the vehicle centreline. To favour the visibility of the light also in sunlight, they have a dark background
- ASMS: Autonomous System Master Switch, it is a switch that allows the voltage supply to the steering and braking actuators

The rulebook imposes to strictly follow the order of the numbered transitions. The default transition is in the state *AS Off*. In this state, the vehicle is completely off and standstill. Selecting the *Manual driving* mode deactivating the *EBS*, keeping the ASMS in off position, and activating the traction system, the vehicle can be driven only by a human driver or pushed externally. Just turning the vehicle off, the transition back to *AS Off* happens. In order to select the *AS Ready* state, the *EBS* must be armed, the ASMS must be set in the "on" position, and then, the traction system turned on. In this state, the braking and steering actuators are supplied by the Low Voltage System (LVS), but the vehicle is still not sensitive to the accelerator pedal position, it is braked to prevent rolling on a slope up to 15%, and the ASSI is yellow continuously. It is possible to return in the previous state just turning off the ASMS and releasing the brakes. The *AS Driving* state is initialized just giving the "Go" signal through the *RES* and waiting 5 seconds of delay. This is the state in which the various Autonomous Missions, still listed in the rulebook, can be performed and it is signalled by the ASSI yellow flashing. As soon as the mission is finished and the vehicle speed is measured equal to zero, the autonomous machine passes into *AS Finished*. In this state, the traction system turns off, the steering actuator too, and an *EBS* manoeuvre is performed to stop the vehicle in a safe position; the ASSI is continuously blue. The *AS Emergency* is the specific state for safety, highlighted by the ASSI flashing in blue. This state is reached from the autonomous states when failures occur with the traction system switched on. Failures can be in the perception system, in the LVS, in the SDC, and in the control algorithms. The *EBS* can be triggered also by pushing the *Stop button* of the *RES transmitter* when unwanted behaviour of the vehicle is noticed by the team. The *RES* triggering translates into *AS Emergency* state also from *AS Finished* state. In the *AS Emergency* state, the vehicle must emit an intermittent sound with an on/off frequency between 1 and 5 Hz, duty cycle of the 50%, sound level between 80 and 90 dBA fast weighting, and duration between 8 and 10 s

after the transition. From *AS Emergency*, it is possible to pass just into *AS Off* when the intermittent sound is "off", the ASMS is switched off, and the brakes are released. The return into *AS Off* is done also from *AS Finished* just turning off the ASMS and releasing the brakes. It is important to underline that the *EBS* release may only be done by manual steps [32].

4.2 Design and testing of the autonomous state machine in Stateflow environment

The first design of the *state machine* (Figure 47) starts with the 7 states imposed by the rulebook. In each state, the outputs of the system are defined with the command "entry" because the actions must happen as soon as the state becomes active.

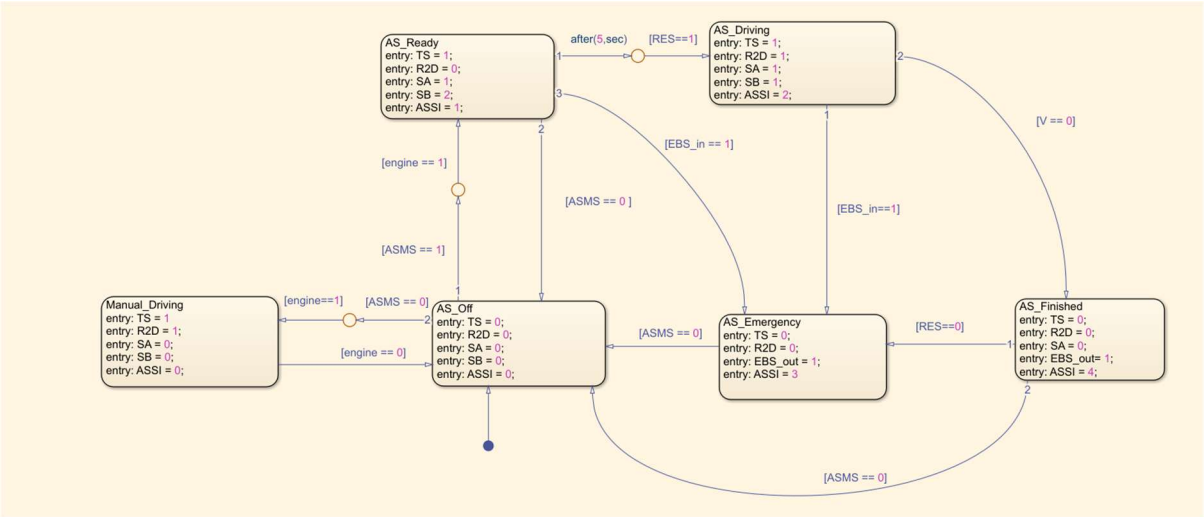


Figure 47. First model of the AS state machine

The input and output variables have the same acronyms of the rulebook, and they are defined into the Symbols Pane (Figure 48) window of the software. The variables have just Boolean values (1 = on and 0 = off) in order to verify just the responsiveness and the behaviour of the model. Only the ASSI variable has 4 values because it is linked to the 4 state that the ASSI LED can assume.

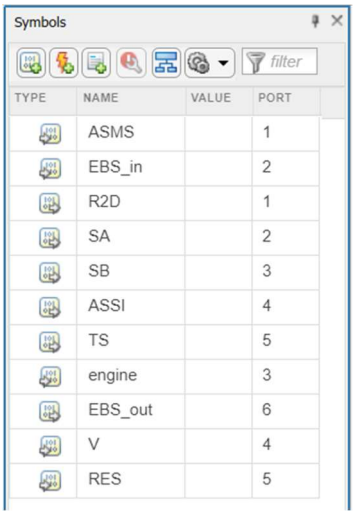


Figure 48. Symbols pane

The simulation is done with infinite stop time to test simultaneously the response of the state machine to the inputs. All the inputs are controlled by toggle switches (Figure 49).

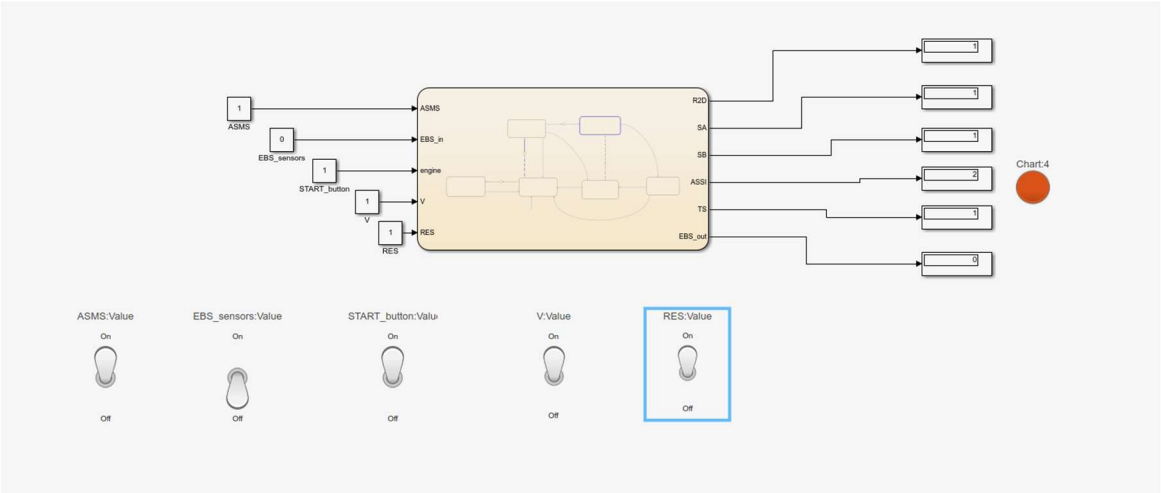


Figure 49. AS state machine model during simulation

The transition to the *Manual Driving* state is execute turning off the ASMS switch and then turning on the START_button switch which simulates the real TS switch on the vehicle to turn on the High Voltage system for traction. In this state, the lamp signalling the ASSI is grey (ASSI = 0). Then, just turning off the START_button, the machine passes to *AS Off*, grey lamp, and it can pass into *AS Ready* turning on the ASMS and afterwards, switching on the START_button; the ASSI lamp becomes yellow (ASSI = 1). The return in the previous state happens just turning off the ASMS_button. From the *AS Ready* state, the machine can

go into *AS Driving* simulating the *RES* signal with the toggle switch *RES*; the ASSI lamp becomes red (ASSI = 2) as it is possible to see in Figure 49 which was taken during a simulation of the model. From *AS Ready* and *AS Driving*, the *AS Emergency* can be triggered by the signal EBS_sensors which simulates any possible error in the vehicle that would trigger the *EBS*, for example errors in the perception system, in the electric circuit or just the *RES Stop button*. The state *AS Finish* is reached when the vehicle velocity is equal to zero and it is highlighted by the ASSI lamp coloured in sky blue (ASSI = 4); in this simple model also the speed is commanded by a switch but in reality it is linked to the measures coming from the Inertial Measurement Unit (IMU). From this last state, the machine can return into *AS Off* switching off the ASMS or it can pass into *AS Emergency* switching off the *RES* toggle switch simulating the press of the *RES Stop button*. *AS Emergency* has the ASSI lamp in dark blue (ASSI = 3). The passage from *AS Emergency* to *AS Off* happens switching off the ASMS.

The order for transitions with two conditions is established using a junction, as for example the 5 seconds delay before the *RES* triggering for the passage to *AS Driving*.

Figure 49 displays the functioning of the designed model: the current state, *AS Driving*, is highlighted by blue boundaries and the red ASSI lamp is also visible. The values of the outputs are displayed, too. The simulation must be paced with simulation time per wall clock second equal to 1 to have real time answers. In this way, the 5 seconds delay of the transition to *AS Driving* can be verified (Figure 50).

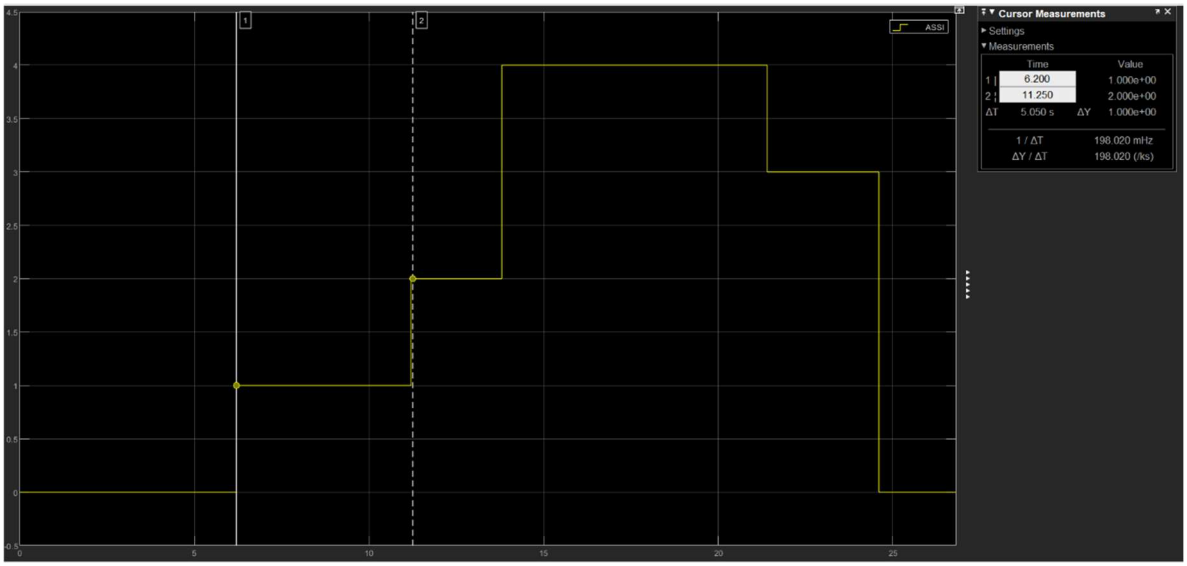


Figure 50. Verification of the 5 seconds delay for *AS Driving* transition

4.3 Integration of the AS state machine with the RES model in Simulink

First refinement of the previous simple model is the integration of the *AS state machine* with the *RES model* explained in Figure 39.

The refinement starts creating the two subsystems (Figure 51). The outputs of the *RES* subsystem are the most significant PDOs: PDO2000 for information about the switches *K2* and *K3*, and on the *Stop button*, PDO2006 for the radio quality. Those outputs become inputs for the AS state machine subsystem allowing the elimination of the *RES* toggle switch.

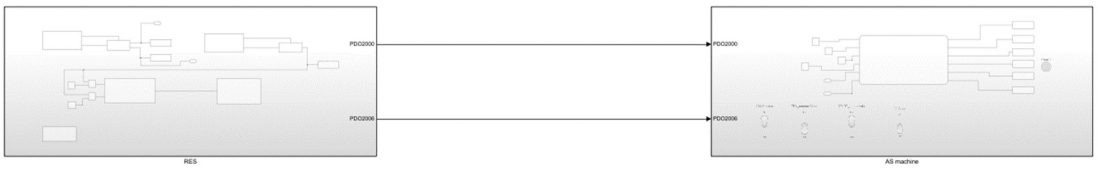


Figure 51. RES and AS state machine models integration

Then, as Figure 52 shows, the transition to *AS Driving* can be signalized by the PDO2000 variable when it becomes equal to 3 or 5. The OR logic is implemented with a | spacing the two conditions. PDO2000 equal to 0 means pressed *Stop button* of the *RES transmitter*, so transition to *AS Emergency*. This transition can happen even when the radio quality, PDP2006, becomes too poor; in this case a value less of the 10% is not allowed because it means that the radio communication with the moving vehicle can be delayed or unable leading to the impossibility of stopping it in case of dangerous situations.

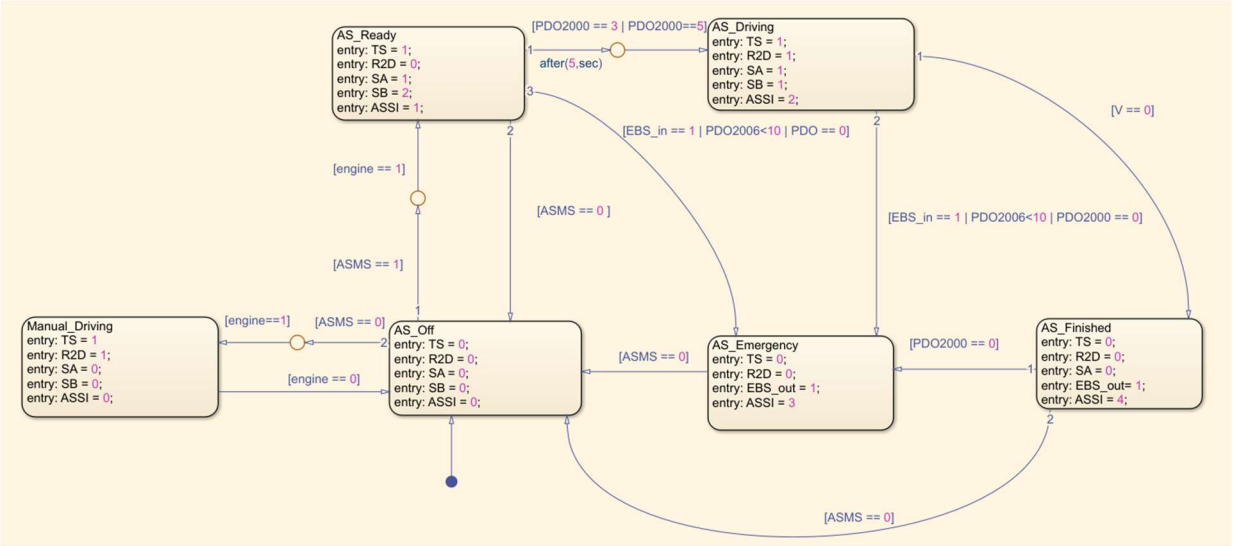


Figure 52. Updated AS state machine with RES inputs

The good and simultaneous behaviour of this model can be seen in the next two figures: the first shows that triggering the toggle switch *K2*, the state machine passes into *AS Driving* state as the red ASSI lamp demonstrates (Figure 53); the last evidences the functioning of the *RES Stop button* to enter in the *AS Emergency* state, in fact the lamp is dark blue and the light inside the button is “off” (Figure 54).

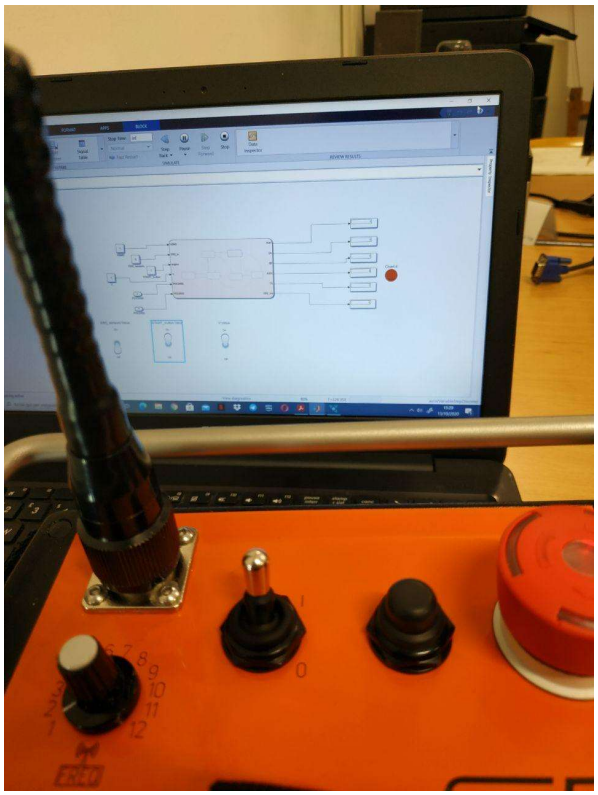


Figure 53. RES providing the Go signal

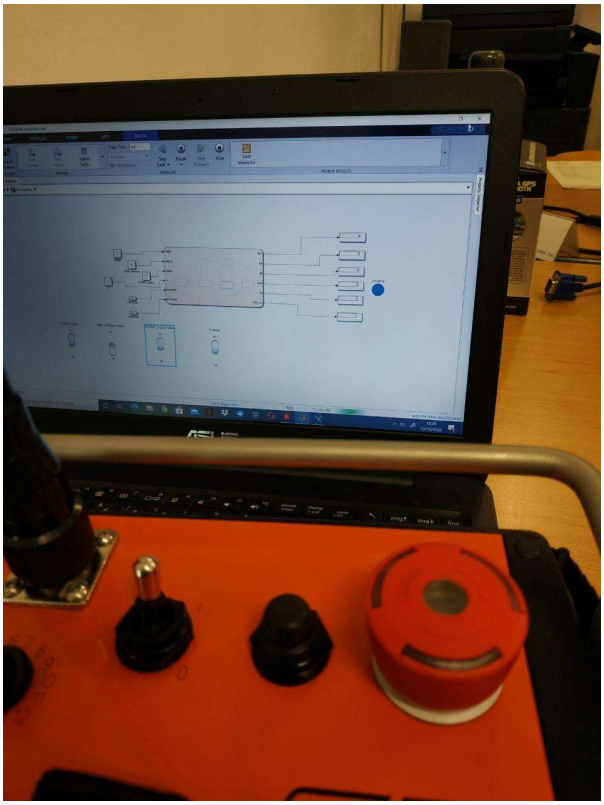


Figure 54. RES providing the Emergency stop

4.4 Integration with EBS hydro-pneumatic and electrical models

Following the idea of validating the *AS state machine* through virtual simulations, giving physical meaning to the input and output variables was the priority. The step-by-step methodology was still followed increasing the simulation complexity only after having verified the success of the first update.

The *EBS* (described in paragraph 3.3) was built on Simulink Simscape (Figure 55). In particular, the *intensifier* is modelled as two *translational mechanical converters* with fixed cases and connected rods; in this way, the pneumatic translational converter represents the air chamber of the intensifier, while the hydraulic one represents the oil chamber. The *canisters* are of course *constant volume chambers* connected to the *gas property block* with data for air. The *pressure regulators* are modelled with a *pressure reducing valve* with the wanted pressure gauge. The *solenoid valves* and the *manual valves* are both modelled with a *3-ways 2-position valve*; thus, the difference is obviously in the command of the valve: the former is commanded by the state machine, the latter by a toggle switch. The geometrical and physical data are memorized thanks to the following MATLAB script:

```
close all
clear all
clc

RefCond.Pressure=206; %[bar]
RefCond.Temperature=25; %[°C]

%% Constant volume chamber
CANISTER.TIPPMAN30009ci_volume=0.000147;                %[m^3]
CANISTER.TIPPMAN30009ci_lenght=0.1365;              %[m]
CANISTER.TIPPMAN30009ci_radius=0.0492;          %[m]
CANISTER.TIPPMAN30009ci_pressure=206;                   % [bar]
CANISTER.TIPPMAN30009ci_diameter_port=0.02059;              % out-port
diameter (5/8") [m]
CANISTER.TIPPMAN30009ci_cross_section_area=pi*CANISTER.TIPPMAN30009ci_dia
meter_port^2/4; %[m^2]
CANISTER.TIPPMAN30009ci_area=24573.16e-6; %[m^2]

%% GAS Properties HA p=206 [bar];T=25 [°]
AIR.specific_gas_constant=0.2871; %[kJ/(kgK)]
AIR.compressibility_factor= 1.107; %[]
AIR.Reference_T_for_gas_properties=298.15; %[K]
AIR.Specific_h_at_reference_T=256; %[kJ/kg]
AIR.Specific_heat_cp=1.338; %[kJ/kg/K]
AIR.dynamic_viscosity=27.77; %[10^-6(Pa*s)]
AIR.thermal_conductivity=26.24; %[mW/m/K]
AIR.density=1.184; %[kg/m^3]

%% Pressure reducing valve #1
PRV.pressure_gauge=55; %[bar];
PRV.nominal_diameter=0.01145;  %internal diameter (1/4")[m]
```

```
PRV.cross_sectional_area=pi*PRV.nominal_diameter^2/4; %[m^2]

%% Pressure reducing valve #2
PRV2.pressure_gauge2=10; %[bar];
PRV2.nominal_diameter2=0.01145;  % internal diameter (1/4")[m]
PRV2.cross_sectional_area2=pi*PRV2.nominal_diameter2^2/4; %[m^2]
%% Pipe GAS1
PIPEGAS.lenght=1.4;     %[m]
PIPEGAS.outside_diameter=0.006;  %external diameter for HPA lines[m]
PIPEGAS.inner_diameter=0.004;  %internal diameter for HPA lines[m]
PIPEGAS.cross_sectional_area=pi*(PIPEGAS.inner_diameter)^2/4; %[m^2]

%% MV 3 WAYS VALVE "T" FEMALE 6700 AIGNEP
MV.inner_diameter=0.01145;  % internal diameter (1/4")[m]
MV.cross_sectional_area=pi*MV.inner_diameter^2/4; %[m^2]

%% Pipe GAS2
PIPEGAS2.lenght=0.08;     %[m]
PIPEGAS2.outside_diameter=0.006;  %external diameter for HPA lines[m]
PIPEGAS2.inner_diameter=0.004;  %internal diameter for HPA lines[m]
PIPEGAS2.cross_sectional_area=pi*(PIPEGAS2.inner_diameter)^2/4; %[m^2]

%% 3/2 NC Solenoid valve 1/4" AIRCOMP
SV.nominal_diameter=0.01145;  % internal diameter (1/4")[m]
SV.cross_sectional_area=pi*SV.nominal_diameter^2/4; %[m^2]

%% Pipe GAS3
PIPEGAS3.lenght=0.03;     %[m]
PIPEGAS3.outside_diameter=0.006;  %external diameter for HPA lines[m]
PIPEGAS3.inner_diameter=0.004;  %internal diameter for HPA lines[m]
PIPEGAS3.cross_sectional_area=pi*(PIPEGAS3.inner_diameter)^2/4; %[m^2]

%% INTENSIFIER gas side
INTENSIFIER_GAS.dead_volume=5480e-9;  %[m^3] calculated directly from the
CAD [mm^3]
INTENSIFIER_GAS.piston=40e-3; %[m]
INTENSIFIER_GAS.rod=18e-3; %[m]
INTENSIFIER_GAS.cross_sectional_area=pi/4*(INTENSIFIER_GAS.piston^2);
% INTENSIFIER_GAS.cross_sectional_area=1144.33e-6;      % [m^2]
INTENSIFIER_GAS.diameter=0.01145;  %internal diameter (1/4")[m]
INTENSIFIER_GAS.mass=0.1;                % mover part mass [kg]
INTENSIFIER_GAS.surface_area=114674.84e-6; %[m^2]

%% INTENSIFIERS oil side
INTENSIFIER_OIL.bore=18e-3; %[m]
INTENSIFIER_OIL.piston_area=pi*INTENSIFIER_OIL.bore^2/4;   %[m^2]
% INTENSIFIER_OIL.dead_volume=30e-9; %[m^3]
INTENSIFIER_OIL.dead_volume=13404e-9; %[m^3]
INTENSIFIER_OIL.stroke=16e-3; %[m]
INTENSIFIER_OIL.mover_mass=0.05; %[kg]
INTENSIFIER_OIL.max_volume=1e-5 %[m^3] from excel calculation volume of
oil of front discs
INTENSIFIER_OIL.max_pressure=38e5; %[Pa]
INTENSIFIER_OIL.pressure=101325; %[Pa]

%% PIPE OIL A04002-TR FRENTUBO
PIPEOIL.internal_diameter=2.5e-3; %[m]
PIPEOIL.passage_area=pi*PIPEOIL.internal_diameter^2/4; %[m]
PIPEOIL.lenght=1; %[m]
```

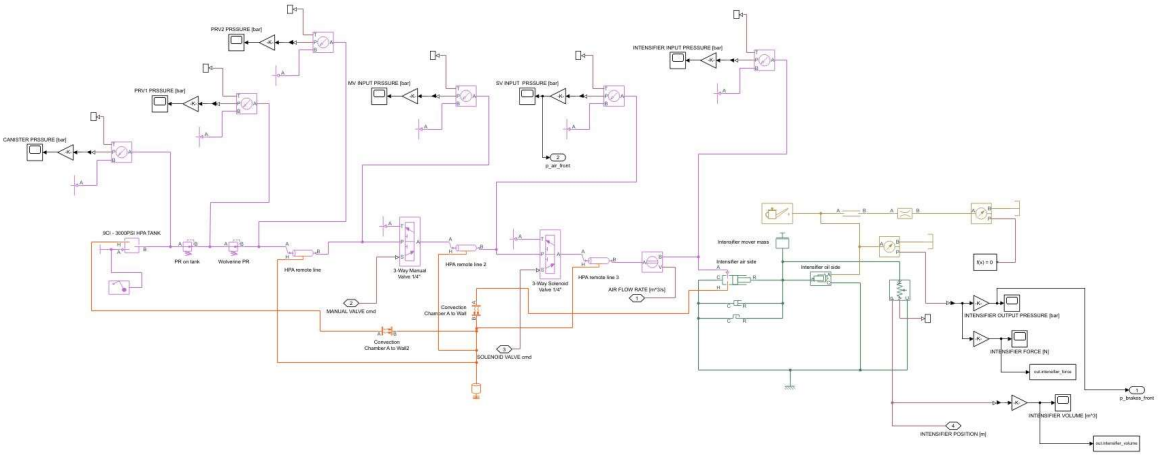


Figure 55. EBS Simscape model

Both the front and rear lines were modelled (Figure 56), and both the mean pneumatic pressure downstream the manual valves and the mean oil pressure are computed to give a feedback to the state machine (Figure 57). In this way, the *state machine* can understand if the *EBS* is *deactivated* (mean pneumatic air is null), *armed* (mean pneumatic air is of 10 bar) or *activated* (mean oil pressure around 49 bar) as imposed by the FS regulation (paragraph 4.1).

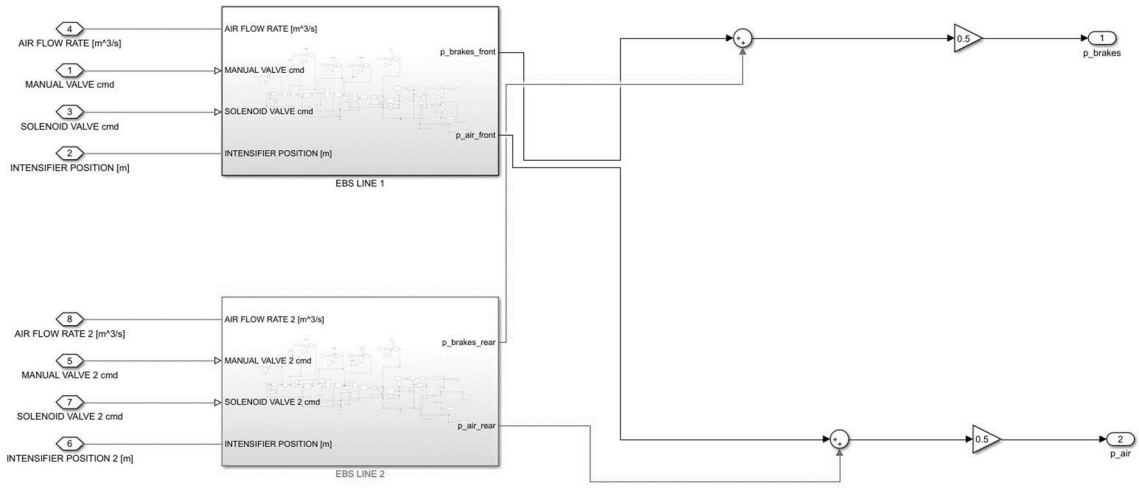


Figure 56. EBS lines

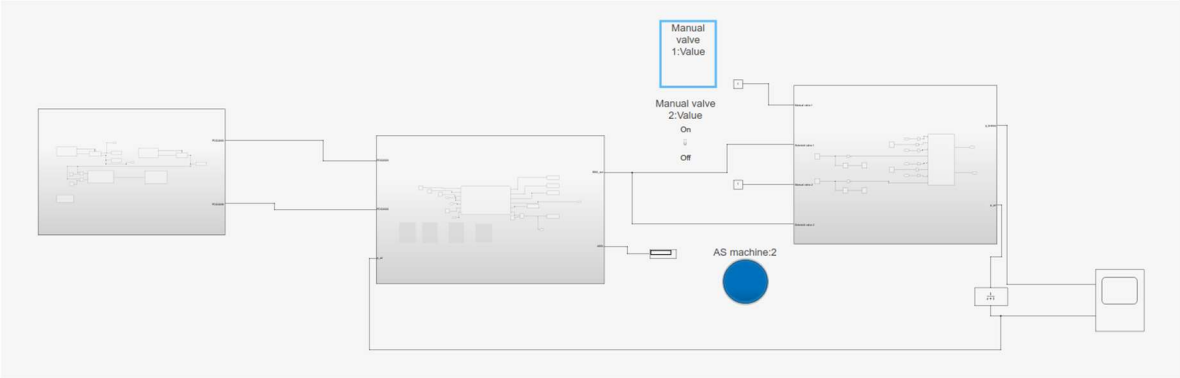


Figure 57. EBS integrated with the AS state machine

From Figure 57, it is possible to notice that the feedback into the state machine needs a first order filter to give a delay for the real-time simulation. The *EBS* is the subsystem on the right. Concerning the *solenoid valves* command, if the variable EBS_out is equal to -1, the EBS is activated; if it is equal to 1, *solenoid valves* lock the passage of the air.

To have this specific values of the EBS_out variable, an artificial mathematical operation is added (Figure 58). The *state machine* was also updated with the new transition conditions about the mean air pressure described before (Figure 59). Those new conditions require a pressure below 1.5 bar to pass into *Manual Driving* since the EBS must be deactivated, so the pressure in the lines should be the atmospheric one, and a pressure above 8 bar to pass into *AS Ready*, since the system can respect the *EBS* manoeuvre requirements.

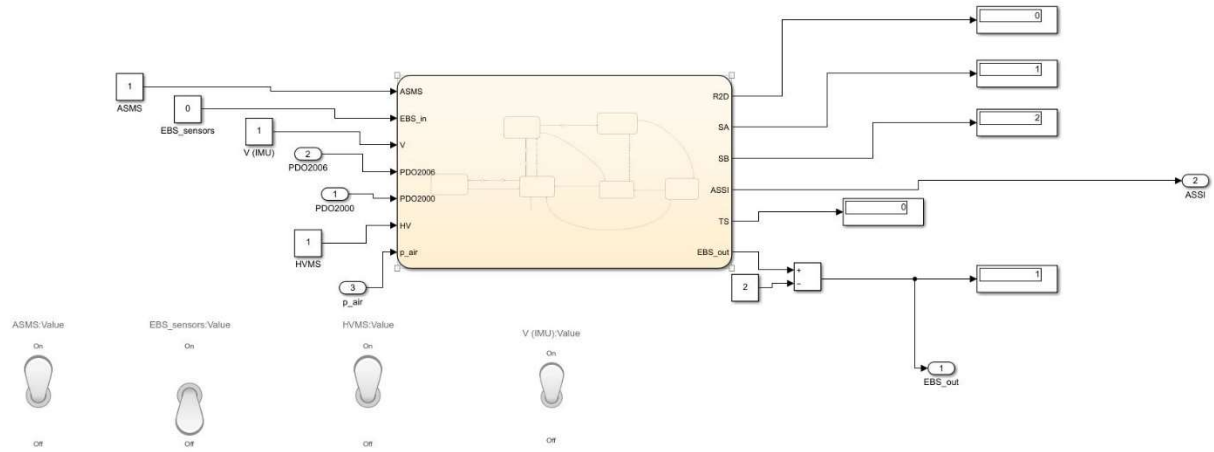


Figure 58. Updated AS state machine for EBS

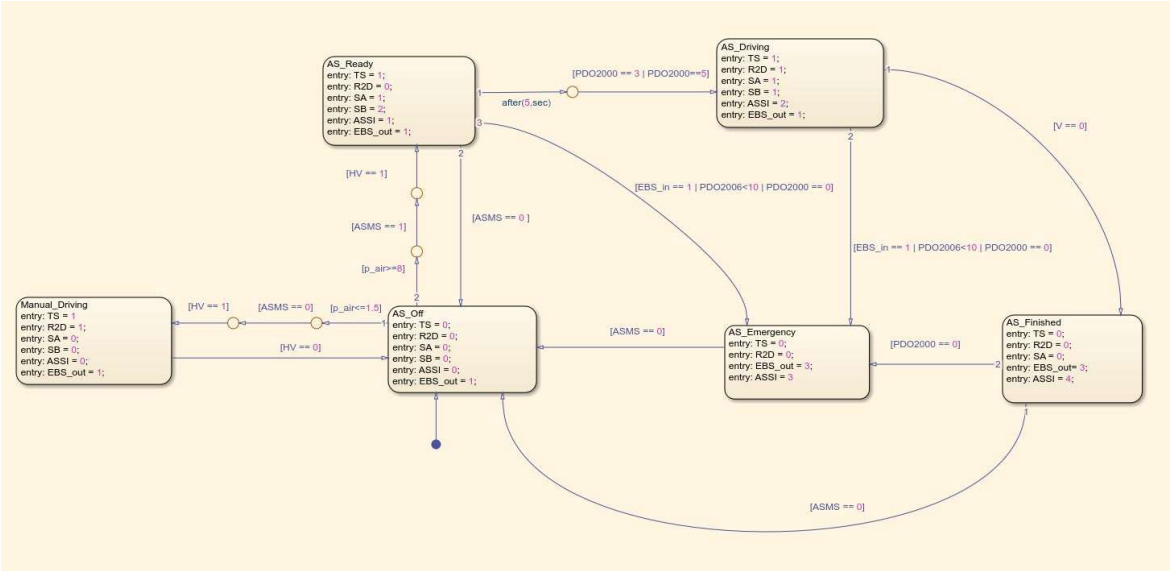


Figure 59. Updated AS chart for EBS

The good response of this implementation is demonstrated by Figure 57 that was taken during a *EBS* simulation triggering the *RES Stop button* and it shows the dark blue ASSI lamp.

In the following Figure 60, there is the possibility to validate the functioning of the model, too. The oil pressure (yellow line) increases as soon as the state machine passes into *AS Emergency* reaching a steady value of around 49 bar. The air pressure (blue line) starts increasing before since it is linked to the activation of the manual valves and it has a steady value of 10 bar. This steady value is not reached immediately because the first order filter has a time constant equal to 1 s for simplicity. The oil pressure (yellow line) has as a dynamic behaviour as a second order damped system excited by step input. The small fluctuations of the value around the steady state are due to numerical errors in the real-time computation that sum-up second after second causing a constant small increase. Those numerical errors are caused by the absence of a physical end to the oil line, which is the disc caliper pushing on the disc in reality. Therefore, the compressibility of the fluids become not negligible, in particular of the air. The axes report the time in seconds on the abscissa (as all the scope in a Simulink model) and the pressures in bar.

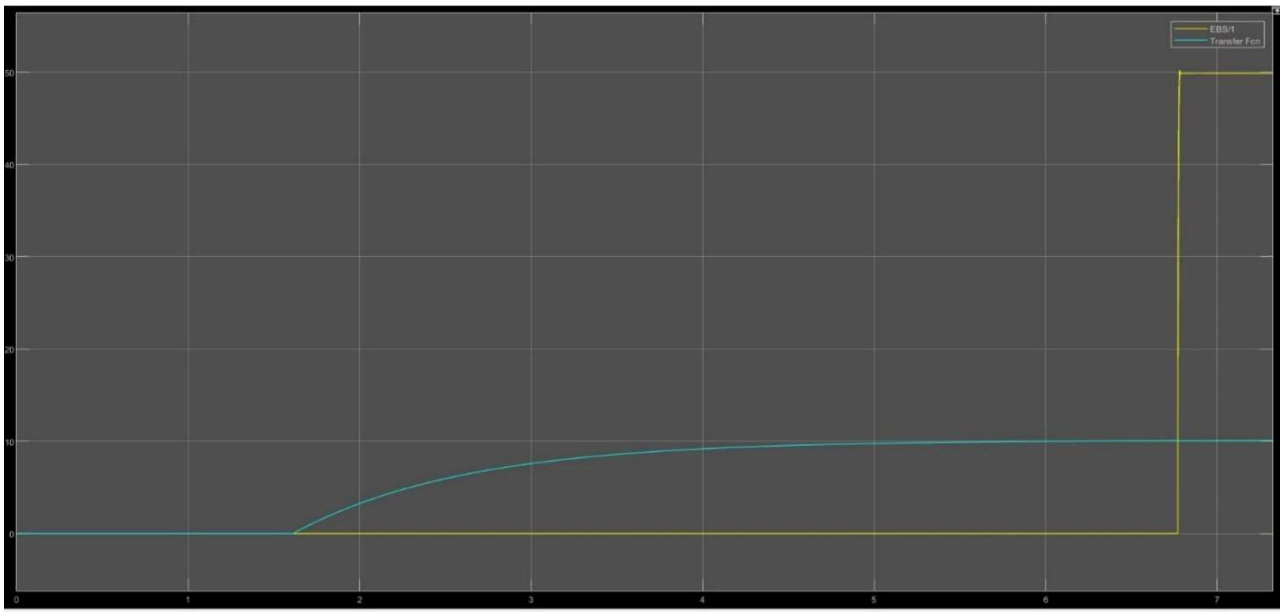


Figure 60. EBS simulation: Air (yellow) and Oil (blue) mean pressures vs time

Then, also the electrical circuit was modelled (Figure 61). For the sake of simplicity, just the most important components for the *EBS* and *state machine* are present. Starting from the Low Voltage Circuit, there are: the *ideal voltage source* of 12 V modelling the battery, the *Low Voltage Master Switch* (LVMS) that activates the Low Voltage Circuit not reported in the model, the *Autonomous System Master Switch* (ASMS) that activates the Autonomous System (AS) part of the circuit, here represented just by the *EBS* part with the relay and a resistance for the solenoid valve electrical absorption. Before the ASMS and still supplied by the Low Voltage source, there is in parallel the Shutdown Circuit with its *High Voltage Master Switch* (HVMS) that is in practice the *Traction System Master Switch* (TSMS), the *RES receiver relay*, and a fictious resistance representing all the current absorption in the circuit as one. Many *current sensors* are in the scheme to give feedback to the *state machine* about the activation of the switches and to command the *EBS relay* and the *solenoid valves*. Moreover, for the feedback into the state machine that must respect a precise transitions order, it is necessary to have the LVMS, the ASMS, and the *EBS* part in parallel. All the resistances have a fictious value of 1 Ω since the scope of the model is just to study the response of the systems and how to manage the *AS state machine* with physical entities. For this reason, the electrical characteristics of all the elements is the given one by Simulink, too.

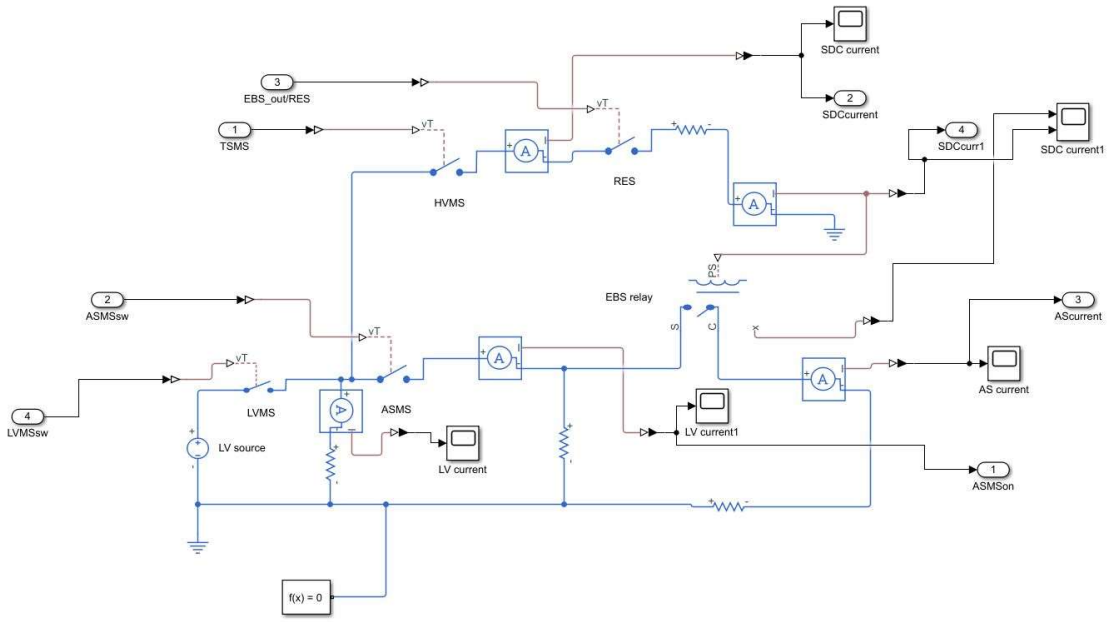


Figure 61. Simplified Low Voltage and Shutdown circuits

The inputs of the LVMS, ASMS, and HVMS are constants commanded by toggle switches as in reality they are physical switches on the body of the vehicle. The *RES* command is the variable EBS_out coming from the state machine since it is equal to 1 when the machine goes into *AS Emergency*. In this way, the *RES* switch can include all the possible openings of the SDC. The outputs are then inputs for the *AS state machine* and *EBS solenoid valves* following the letters, as visible in Figure 62.

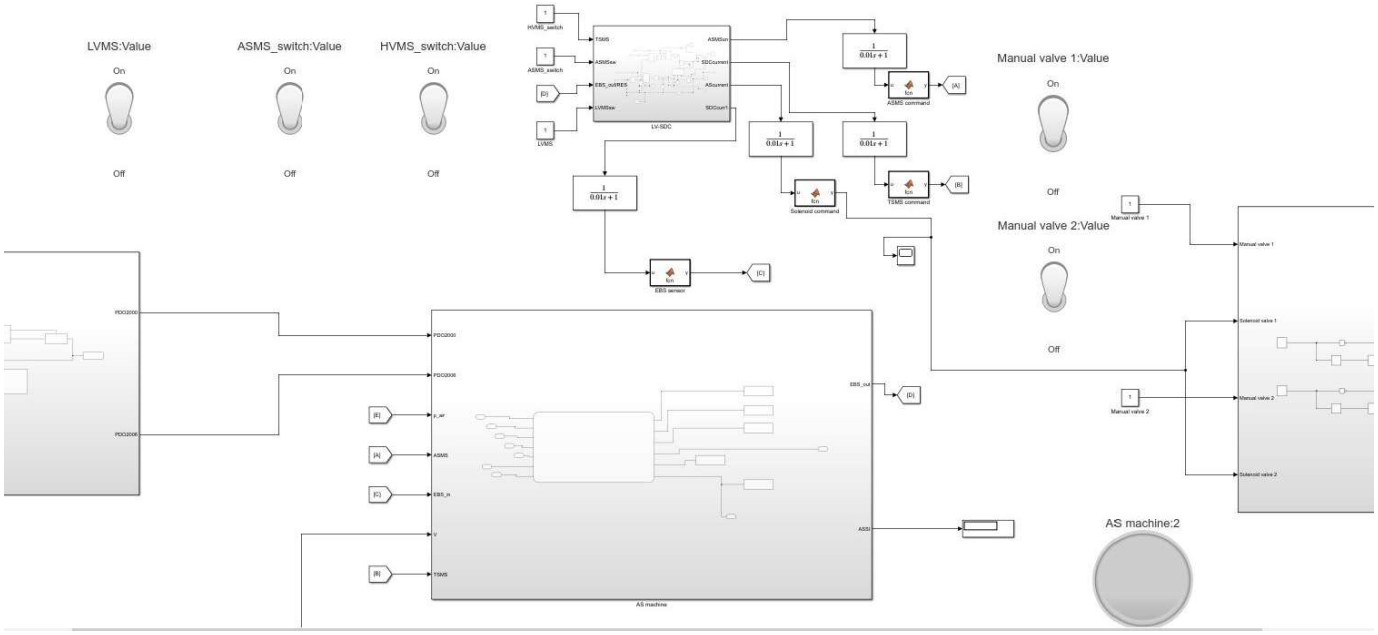


Figure 62. AS State Machine links with the electrical circuits

Figure 62 shows the necessities of other first order filters for the real-time simulation and many functions since the state machine accepts Boolean values while in the circuit, there are analogical values. To reduce the delay introduced by the first order filter, the time constant was imposed equal to 0.01 s. The "ASMS command" and the "TSMS command" functions are the same since the output variable (ASMS/TSMS) is 1 if there is current flowing in the respective loop:

```
function ASMS = fcn(ASMSon)
if ASMSon>1
    ASMS=1;
else
    ASMS=0;
end
end
```

The "Solenoid command" function is different since it must give 1 if there is no current flowing in the *EBS* loop of the Low Voltage circuit:

```
function solCMD = fcn(AScurrent)
if AScurrent<=0.1
    solCMD=1;
else
    solCMD=-1;
end
end
```

The inequality with 0.1 is due to the prevision of some tolerance.

The current flowing in the SDC also gives a feedback with the variable EBS_in since in reality the *EBS* is activated by every failure in this circuit. The function to translate in a Boolean command the absence of current is the following:

```
function EBS_in = fcn(SDCcurr1)
if SDCcurr1<=0.1
    EBS_in=1;
else
    EBS_in=0;
end
end
```

The value of 0.1 has the same meaning of the previous function.

Validation of this update in the model is given by the following plots. The abscissa is always the time in second, while the y-axis can represent the current in Ampere or the pressure in bar depending on the plotted variables.

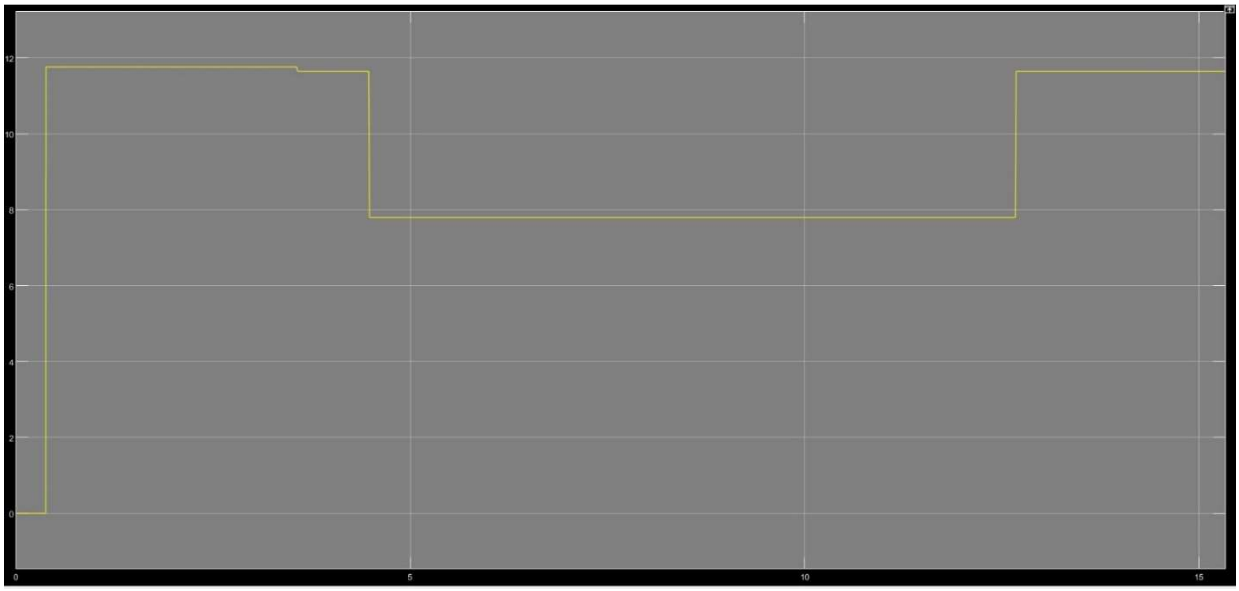


Figure 63. Low Voltage circuit current vs time

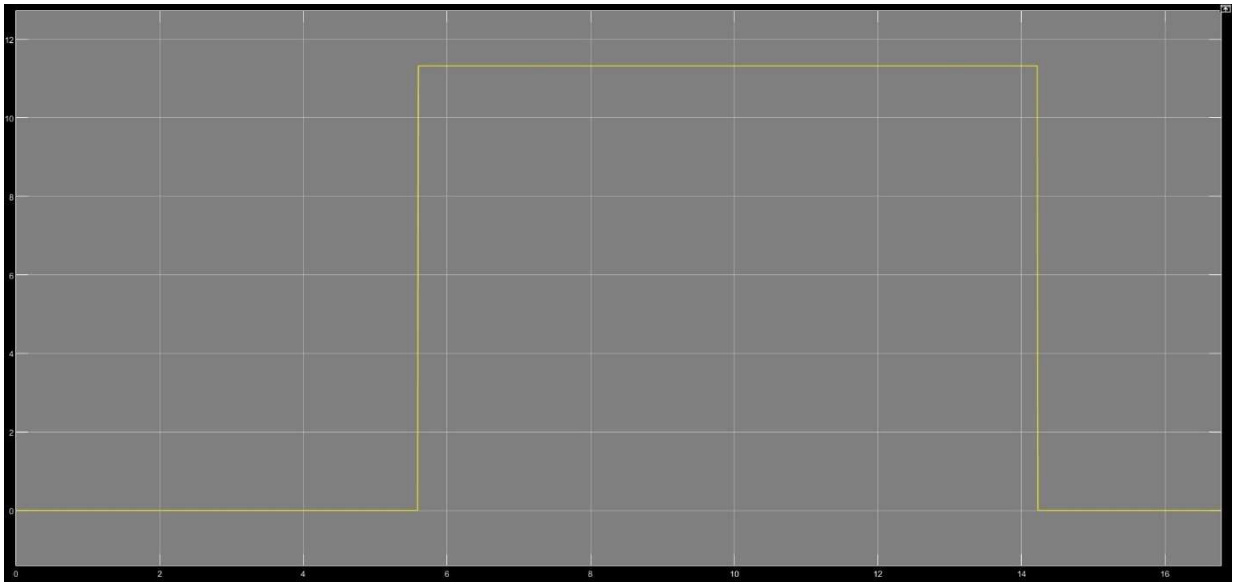


Figure 64. *Shutdown Circuit current vs time*

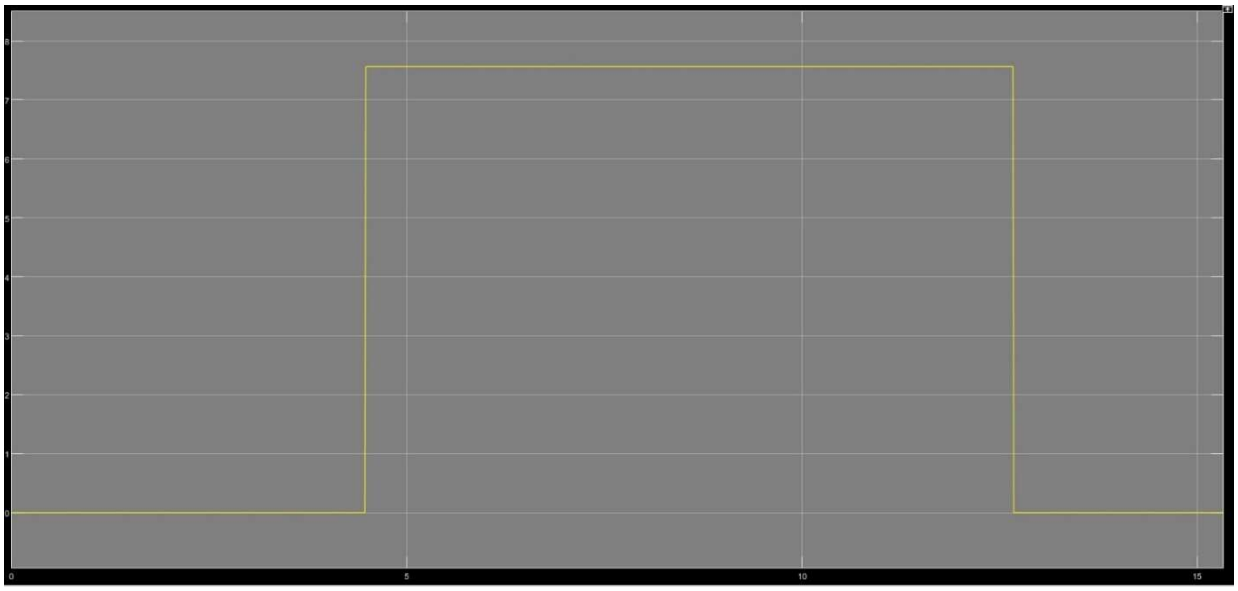


Figure 65. Autonomous System current vs time

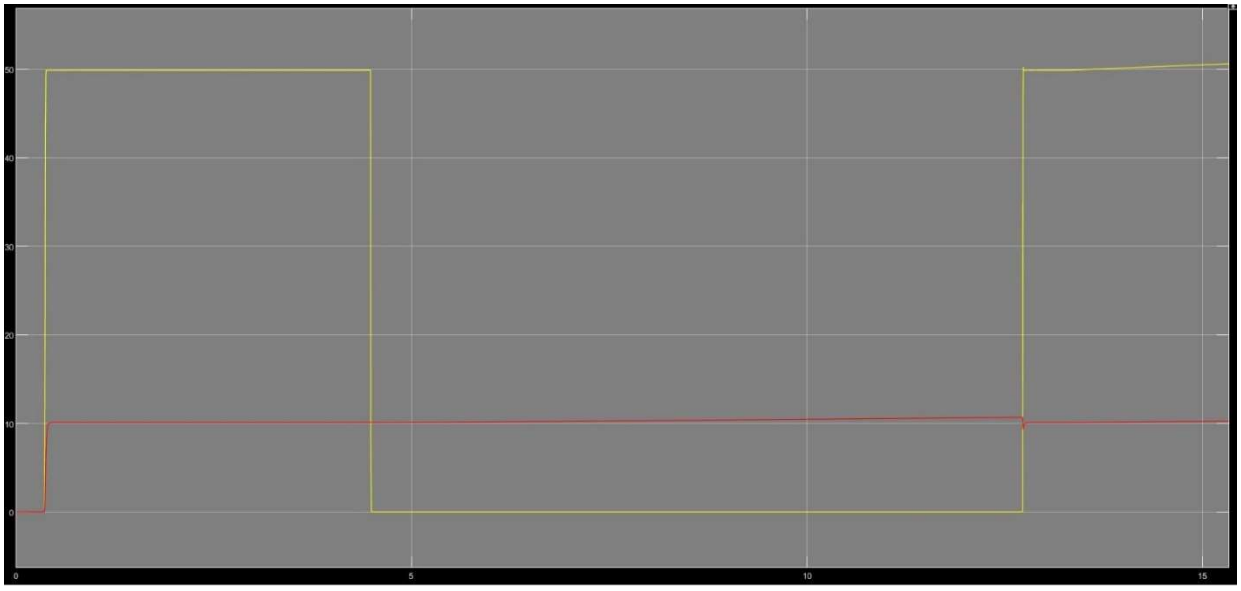


Figure 66. Electric circuit simulation: Air (red) and Oil (yellow) mean pressures vs time

The simulation was executed as a real Autonomous driving should be following the steps of the theoretical *state machine* in paragraph 4.1. Firstly, the LVMS is switched on. Then, the manual valves are activated, so the air pressure build-up can be seen. The oil pressure grows, too, since the solenoid valves are open due to the absence of current caused by the ASMS off. So, the ASMS and the HVMS are activated, as Figure 63 shows with the current drops. As soon as the SDC has current (Figure 64), also the AS current starts flowing due to the

closure of the relay. So, the solenoid valves commutate in the closed state and the oil pressure drops to zero. In this situation, the *state machine* is in *AS Ready* and a *Go signal* from the *RES* initialise the transition to *AS Driving*. Finally, the *Stop button* of the *RES* is triggered, so the SDC opens, consequentially also the AS part of the Low Voltage circuit opens triggering the *solenoid valves* in open state. This actions are demonstrated by the last parts of Figures 64, 65, and 66. Investigating better the plots on the software, it is possible to evidence a 40 ms of delay between the SDC current drop and the oil pressure build-up, and this is far away the 200 ms of maximum *EBS* activation time required in the FS regulation.

4.5 Further refinements: Missions, Ready-to-drive, EBS check-up sequence

Besides the autonomous states, the FS rulebook defines the *Autonomous Missions*. Those must be at least the following:

- Manual driving
- Inspection
- Acceleration
- Skidpad
- Autocross
- Trackdrive
- EBS test

The first one is just the selection of the M*anual driving state*. The *Inspection* mission must be used during the technical inspection when the vehicle is jacked up with removed wheels; the drivetrain must slowly spin, and the steering system must follow a sine wave for an overall duration of 25÷30 seconds. *EBS test* is still part of the technical inspection and it evaluates the response and the rule-compliance of the *EBS* when triggering the *RES*. All the other missions respond to the dynamic events of the competition precisely described in the rulebook. *Acceleration* evaluates the acceleration performance on a straight path. *Skidpad* is a constant-steering test using two circular paths to test both the steering directions. *Autocross* evaluates the handling performance on a specific track. *Trackdrive* substitutes the Endurance test for manual driving vehicles and it challenges the driverless vehicle in a closed-loop track [32].

The selected mission must be indicated by the *Autonomous Mission Indicator* (AMI) [32].

Missions are fundamental for the driverless car because each one has its own control strategy of both the traction system and the steering actuator and also its own path planning strategy.

The implementation of the missions in the *AS state machine* happens in the *AS Driving* state where a specific sub-state for each mission can be written (Figure 67). The default mission was randomly imposed as the *Acceleration* one, but the prior selection ensures immediacy of the response. In fact, the mission selection must be done during the *AS Off* state and it is a needed transition condition, as Figure 67 shows. *Manual driving* is the only mission that allows to pass into *Manual driving* state, while the other missions allow to pass into *AS Ready* following an exclusive (OR) logic. The missions are numbered in order to give consistency to the variable *mission*, imposed into the model thanks to the *edit block* which allows to write the value of the variable during the simulation:

- 0 = Manual driving
- 1 = Acceleration
- 2 = Skidpad
- 3 = Autocross
- 4 = Trackdrive
- 5 = EBS test
- 6 = Inspection

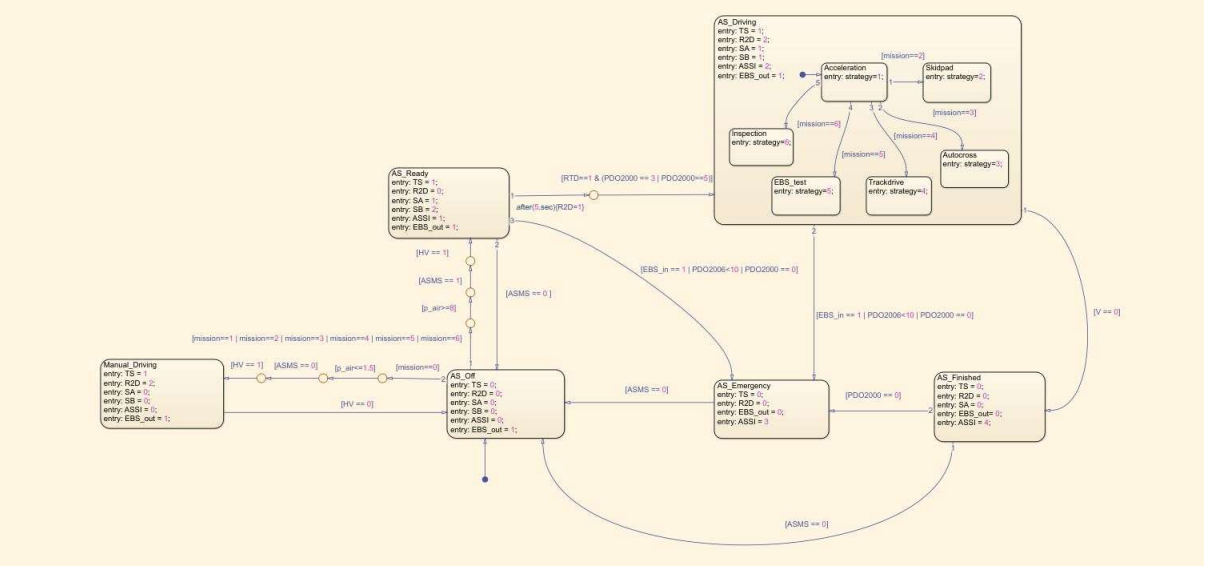


Figure 67. State machine with missions

Once a mission is active in *AS driving*, the variable *strategy* is an output to activate the specific control logic.

Another improvement is the implementation of the *Ready-to-drive* function. *Ready-to-drive* is the vehicle mode during which the motors can respond to the Acceleration Pedal Position Sensor (APPS). The rulebook does not define strictly rules for the activation of this vehicle mode after turning on the tractive system; the only clear restriction is the engage of the mechanical brakes while the activation procedure (for example by a button on the dashboard) is executed by the driver [32].

Considering the driverless competition, the *Ready-to-drive* procedure must be done autonomously during the transition from *AS Ready* to *AS Driving*. So, as also the FSG EBS Reference Guide suggests, a MOSFET switch downstream the solenoid valve is added to control the mechanical brakes through the central control unit [44]. Thus, the electrical scheme was modified to distinguish the front and rear lines in parallel (Figure 68).

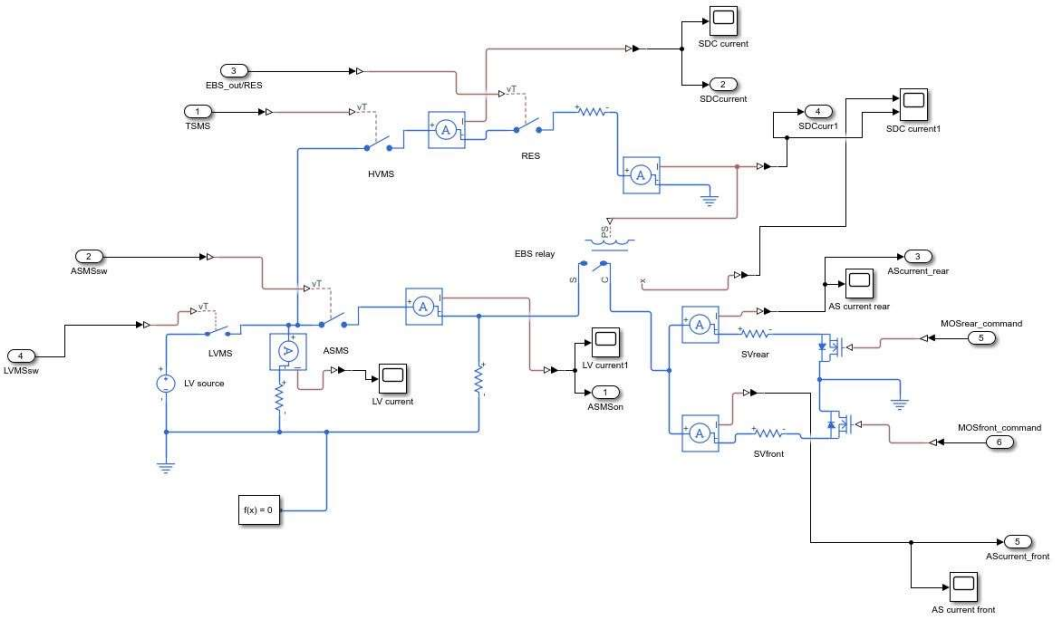


Figure 68. Electrical scheme with the two parallel lines

In order to implement the *Ready-to-drive* procedure in the state machine, the values of the variable R2D were changed: 0 = “off”, 1 = “start procedure”, and 2 = “on”. The variable R2D becomes 1 thanks to a transition action. Moreover, another variable (RTD) was added for the verification of the procedure end during the last transition condition (Figure 69).

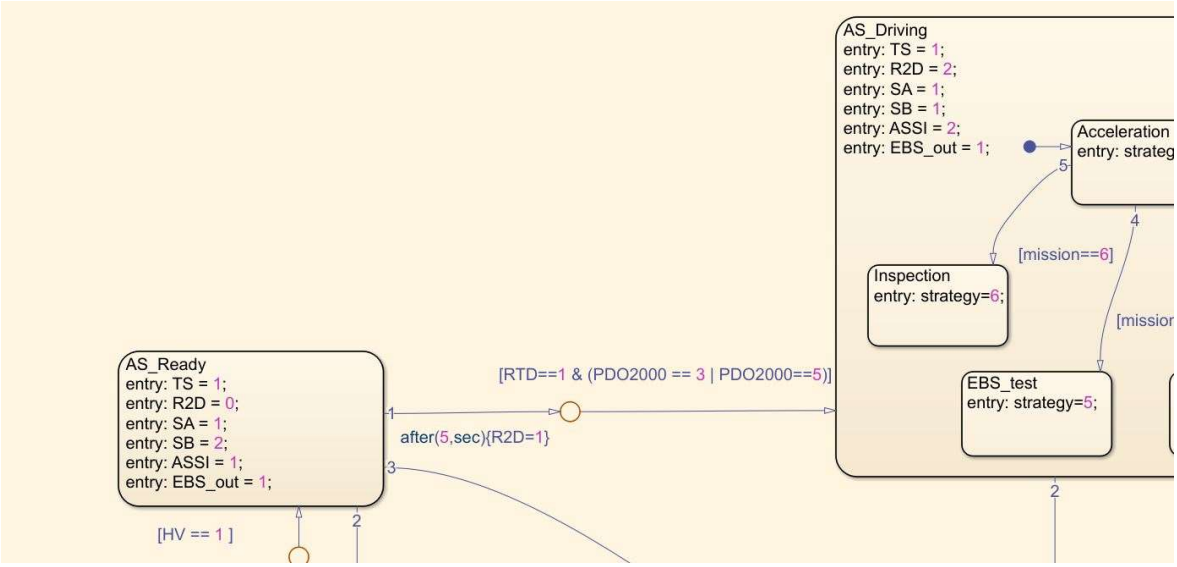


Figure 69. Ready-to-drive procedure inside the AS state machine

The variable R2D is an output of the chart (Figure 70) that goes inside a function to command the MOSFET:

```
function MOS = fcn(R2D)
if R2D==1
    MOS=0.1;
else
    MOS=4.5;
end
end
```

The values of MOS are justified by the standard threshold voltage pre-imposed by Simscape of 2 V. So, MOS less than 2 means switch in off position, and consequentially, mechanical brakes activated.

In Figure 70, the links between the output R2D of the state machine and the electrical circuit can be seen. The command of the solenoid valves can be also noticed which follows the previous function but splitted into the two lines.

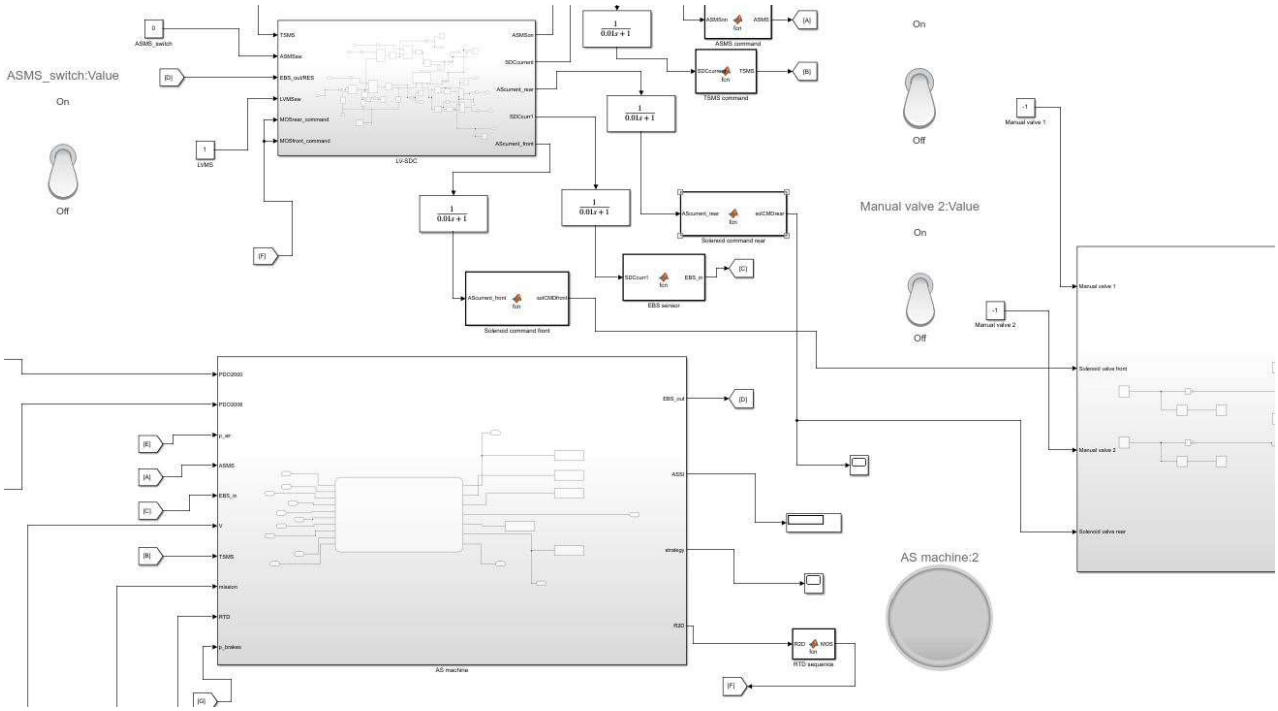


Figure 70. Part of the overall Simulink model to highlight the R2D links

Once activated the brakes, the feedback from the mean oil pressure is used to start the *R2D* procedure (Figure 71). As all the feedback in the model, the oil pressure needs to be delayed with a very small time constant (0.01 seconds).

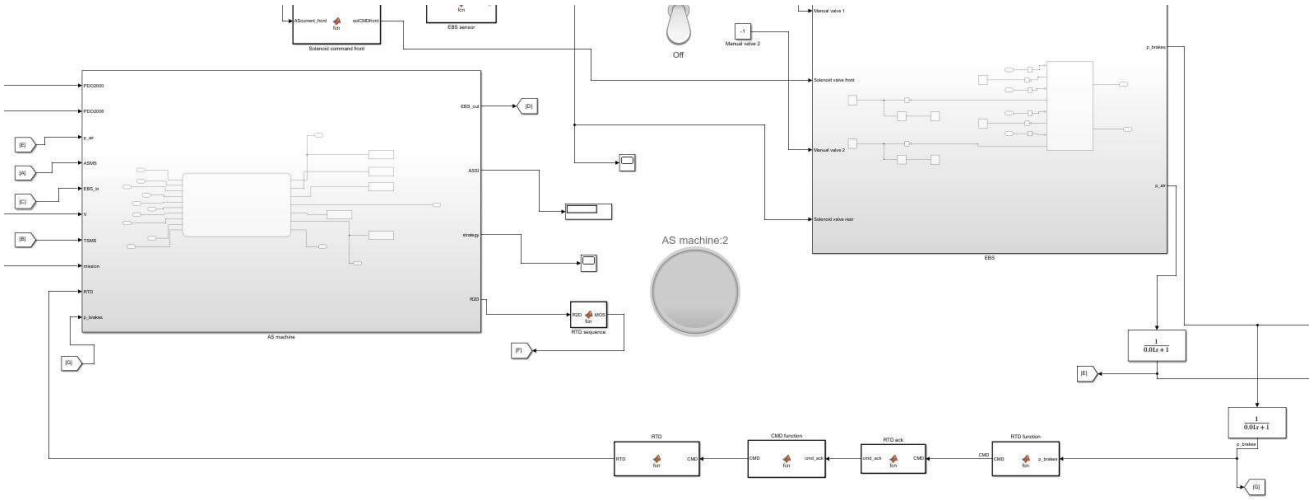


Figure 71. R2D procedure and feedback

To better understand the idea behind the 4 consecutive functions before the feedback in the *state machine* (Figure 72), at least the part the actual SC19 *Ready-to-drive algorithm* in which those functions will be inserted must be presented:

```
%First push, send acknowledge
if CMD == 1 && rtd==0 && ctor_en==0
    cmd_ack = 1;
    TxEn = 1;
    ctor_en = 1;
    rtd=0;

elseif CMD == 2 && rtd==0 && ctor_en == 1 && CTOR_STS==8    %Second push,
send acknowledge, RTD ON

    cmd_ack = 2;
    TxEn = 1;
    rtd=1;
end
```

So, the driver must push two times the *R2D button* on the dashboard to activate this function, always keeping the brake pedal pushed.

The first function block is useful to translate the oil pressure signal into a CMD signal for the already existing *R2D algorithm*:

```
function CMD = fcn(p_brakes)
if p_brakes>=40
    CMD=1;
else
    CMD=0;
end
```

The second block is a fictious function simulating the first push:

```
function cmd_ack = fcn(CMD)
if CMD == 1
    cmd_ack=1;
else
    cmd_ack=0;
end
end
```

The acknowledgement data is then used to send the second CMD signal:

```
function CMD = fcn(cmd_ack)
if cmd_ack==1
    CMD=2;
else
    CMD=1;
end
```

Eventually, the last block simulates the second push part of the original algorithm:

```
function RTD = fcn(CMD)
if CMD==2
```

```
    RTD=1;
else
    RTD=0;
end
end
```

The value RTD = 1 is the transition condition needed to pass into *AS Driving* simultaneously with the *Go signal*.

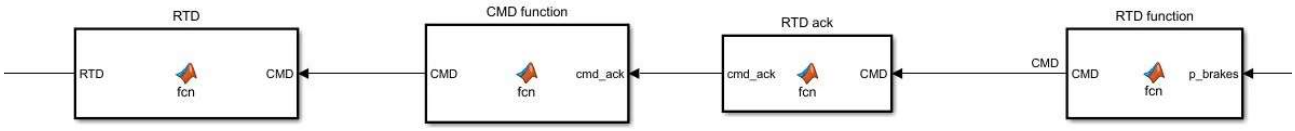


Figure 72. Ready-to-drive simulation model

The good model compiling is demonstrated by the plot in Figure 73, where the mean air (red) and oil (yellow) pressures in bar versus time in seconds are reported. The first mean pressures increase is due to the activation of the manual valves before giving power to the solenoid valves. The second peak of oil pressure is caused by the *Ready-to-drive* procedure. The last increase in oil pressures stands for the *EBS* activation at the *end of the autonomous mission*. Lastly, the two releases of pressures follow the deactivation of the manual valves.

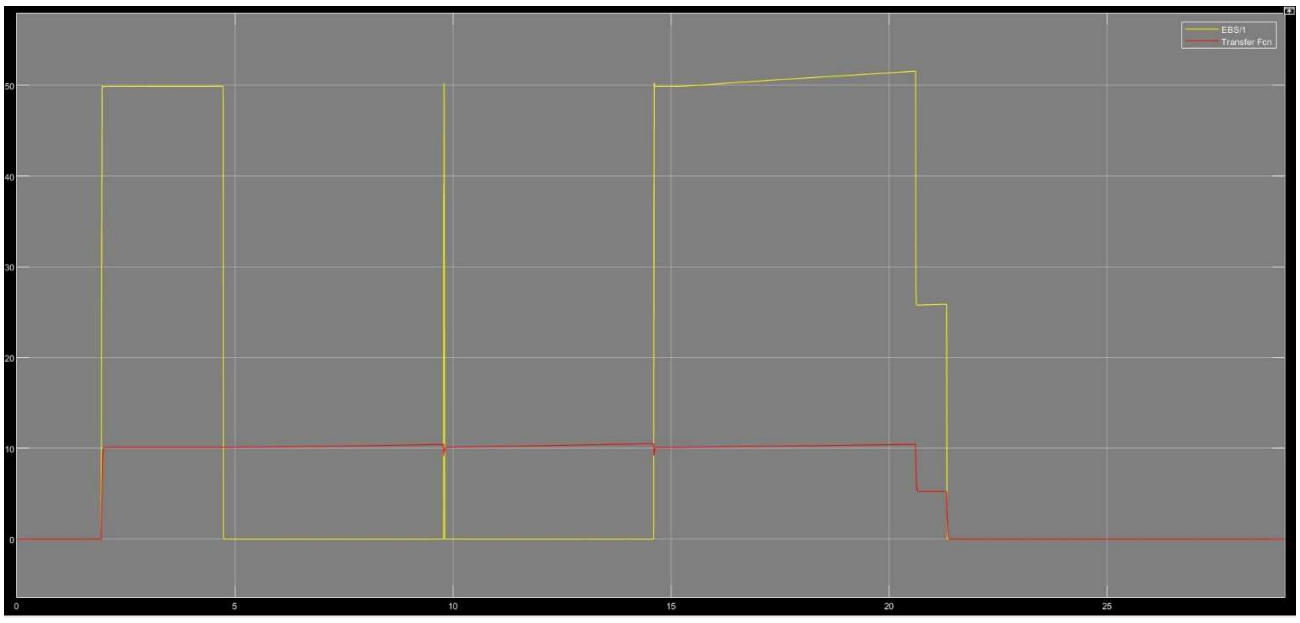


Figure 73. R2D simulation: Air (red) and Oil (yellow) mean pressures vs time

Last update of the thesis model is the implementation of the *EBS check-up sequence* in the state machine. This sequence is prescribed by the FS rulebook to verify the possibility of

building-up pressure in the brakes before passing into *AS Ready* [32]. The sequence is necessary to discover all kinds of failure not detectable without triggering the *EBS*, for example due to failures in the assembling of the components. It is requested also to verify the redundancy of the system and the functioning of the Supervisor [44]. Similarly to the *R2D sequence*, the MOSFETs of the two lines are used to activate and deactivate the *EBS*.

The first modifications are in the *state machine* chart (Figure 74). The activation of the ASMS is followed by a transition action, *AS_close_SDC* = 1, that enables to activate the TSMS. This action is a simplification of the *Watchdog verification* for the Supervisor that will be implemented successively. Once the TSMS is activated, another transition action, *EBScheck* = 1, initializes the check-up function. Then, the last transition to *AS Ready* follows the condition of a successful EBS check-up, *EBS_result* = 1.

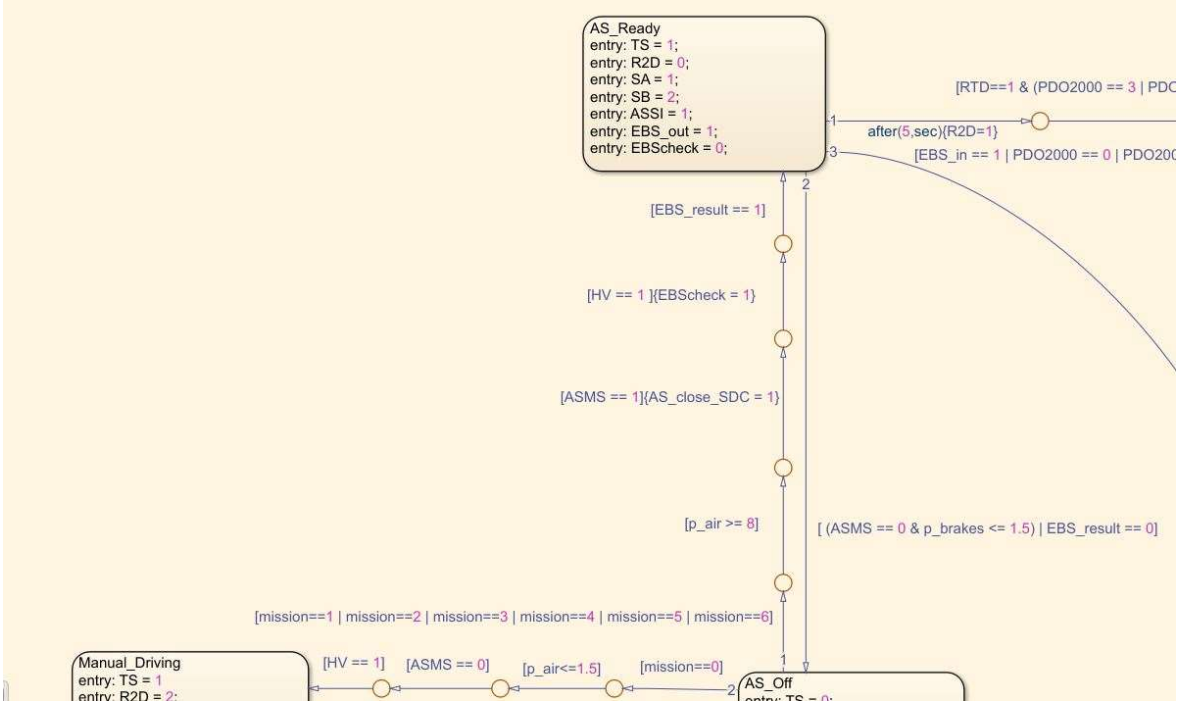


Figure 74. Chart modifications for EBS check-up sequence

The *EBS check-up function* verifies the functioning of the EBS. Firstly, the overall system is verified in building-up pressure. In order to give a precise order, an internal variable *ack* is defined and it increments of 1 as the verification step is completed. As *ack* = 1, the front brakes are controlled. Then, only the rear brakes are controlled, and finally, the feedback to the *state machine* is sent. The MATLAB function computing this routine is the following:

```
function [EBS_result , MOSfront_comm , MOSrear_comm]= fcn(EBScheck ,
p_brakes_front , p_brakes_rear)

persistent ack
if isempty(ack)
    ack = 0;
end

if EBScheck == 1
    MOSfront_comm=1;
    MOSrear_comm=1;
else
    MOSfront_comm=0;
    MOSrear_comm=0;
    EBS_result=0;
end

if (p_brakes_front + p_brakes_rear) >= 80 && EBScheck == 1
    ack = 1;
end

if ack == 1
    MOSfront_comm=1;
    MOSrear_comm=0;
end

if p_brakes_front >= 40 && p_brakes_rear <= 1.5 && EBScheck == 1
    ack = 2;
end

if ack == 2
    MOSfront_comm=0;
    MOSrear_comm=1;
end

if p_brakes_rear >= 40 && p_brakes_front <= 1.5 && EBScheck == 1
    ack = 3;
end

if ack ==3
    MOSfront_comm=0;
    MOSrear_comm=0;
    EBS_result=1;
else
    EBS_result=0;
end

end
```

The input data are *EBScheck* from the state machine and the pressures of the front line and rear line of the braking system. The output data are the commands for the MOSFETs and the feedback to the state machine *EBS_result* (Figure 75). All the feedback data requires a small delay for compiling, as discussed in the other paragraphs. Since the MOSFETs can be controlled both by the *EBS check-up function* and the *R2D sequence*, a function is required to implement a OR logic:

```
function MOSfront_comm = fcn(F , I)

if I == 1 || F == 1;
    MOSfront_comm=0.1;
else
    MOSfront_comm=4.5;
end

end
```

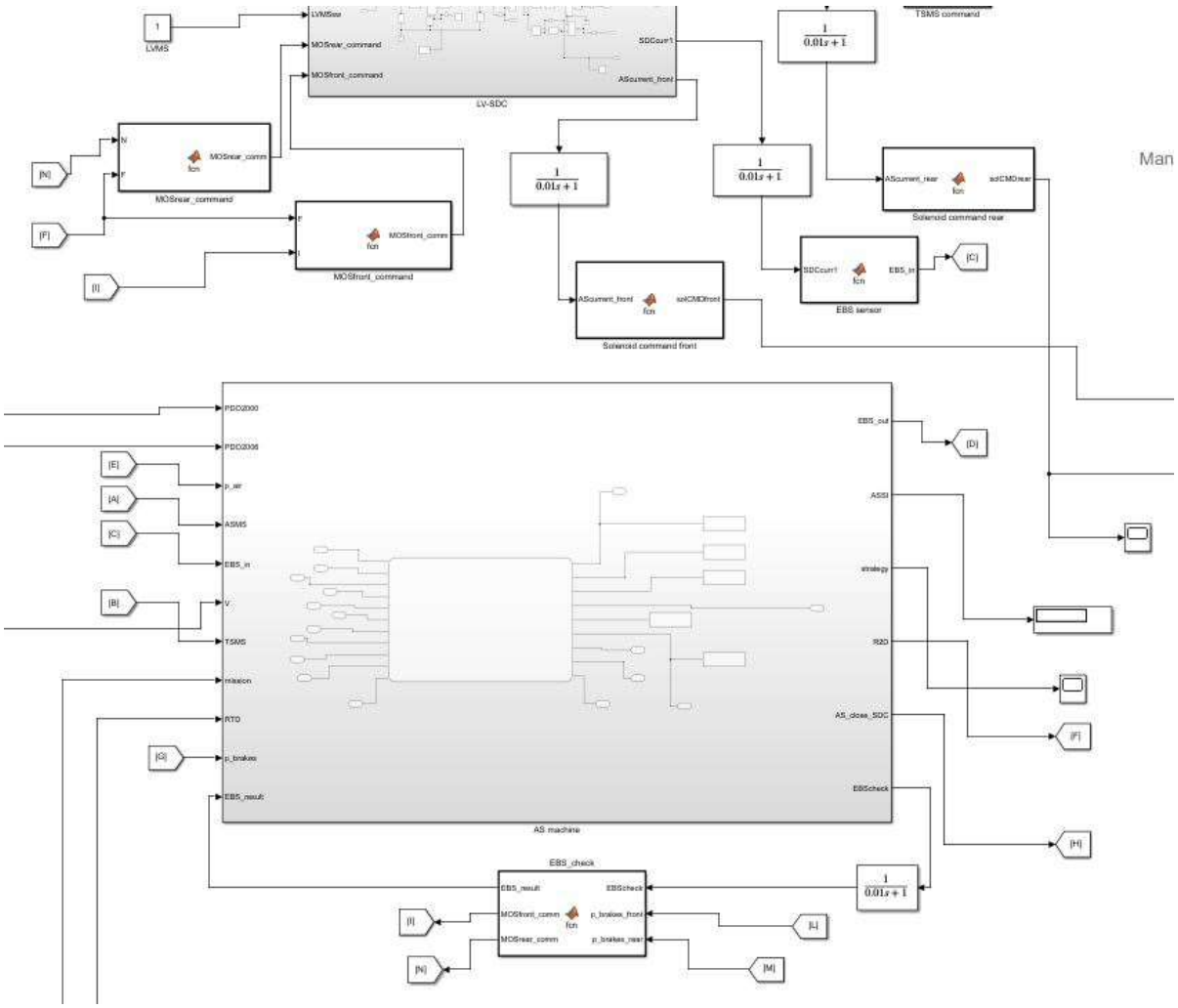


Figure 75. EBS check-up function feedback and MOSFETs commands

A final complete simulation of typical autonomous routine was done to validate this last model. The plot of the mean pressures is useful to understand the behaviour of the *AS state machine* (Figure 76). The two initial steps in the air pressure (red line) are caused by the activation of the manual valves. Since the SDC is still open due to the TSMS open, the oil pressure (yellow line) increases, too. Once turned on the ASMS and the TSMS, the oil

pressure falls immediately to zero and then, the *EBS check-up* starts with one peak to 49 bar due to the verification of the overall system, then, with a reduction to around 25 bar due to the verification of the front brakes, and then, a peak always around 25 bar due to the control of the rear brakes. After this check, the state machine is in *AS Ready*, and after 5 seconds the *Ready-to-drive sequence* happens as it can be seen with the new activation of the *EBS*. As soon as the *RES Go signal* is sent, the oil pressure is released, and the mission starts in *AS Driving*. At the end of the mission, the *EBS* is triggered and the two final releasing steps of the oil pressure are caused by the discharge of the air pressure in the *EBS* lines thanks to the manual valves.

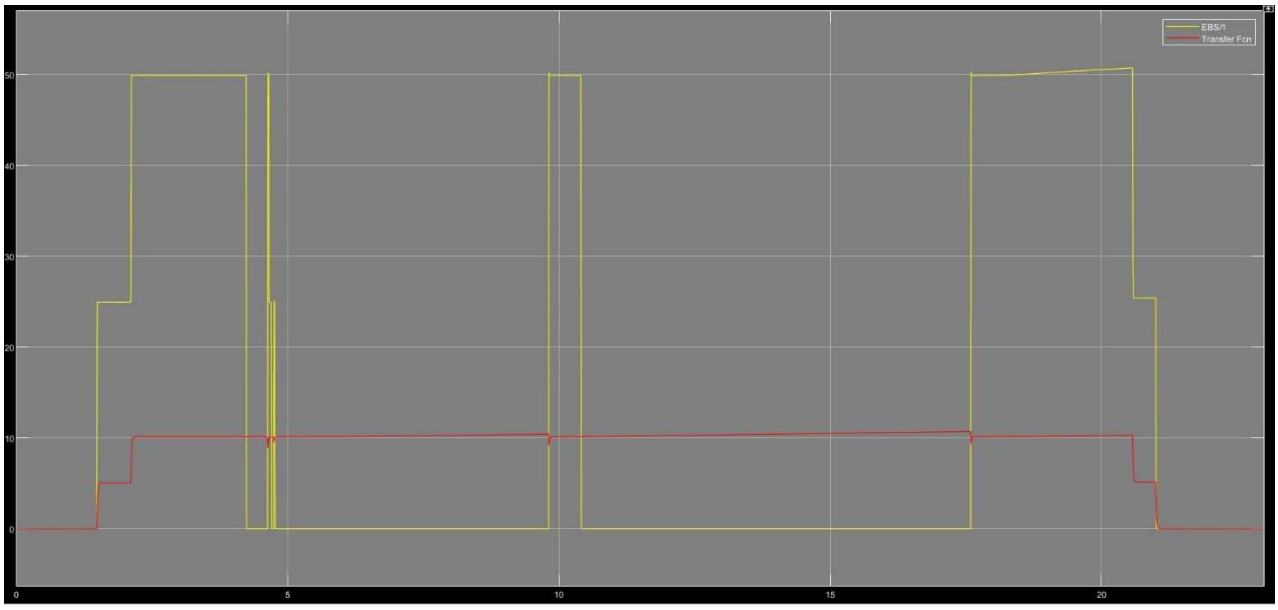


Figure 76. Complete simulation: Air (red) and Oil (yellow) mean pressures vs time

Chapter 5 – Conclusions and future steps

As the thesis body is divided into two macro-areas of the work, the conclusions must be disclosed following those two, too.

About the *RES*, the testing phase was successful both as single subsystem and as integrated subsystem with the *EBS*. Next steps will be concentrated in the mounting inside the vehicle and tuning the overall *EBS* by reproducing a real *EBS test* as described in the rulebook. Moreover, the prepared DBC file will be very useful for the implementation of the *RES communication* with the central control unit.

Concerning the *AS state machine*, all the models compiled in Simulink environment, and the progressive refinements had the goal of giving more and more physical sense to the inputs and outputs. Furthermore, the Simulink models gave the possibility to understand how to pilot some electronic components before purchasing them. As it is possible to notice in the discussion of Chapter 4, some variables inside the states have still no physical meaning. However a preliminary study has been already conducted:

- The variable *TS* (traction system) must be commanded by the TSMS as in the simulated model, but during the transition from *AS Ready* to *AS Off* and from *AS Driving* to *AS Finished*, the autonomous machine should be able to deactivate the traction system without any external command. Thus, next step will be to insert a normally-open relay in the SDC that will be piloted by the state machine output TS thanks to the connection of the control unit with a MOSFET to give voltage to the relay coil. This solution is already used in the existing SDC, so the work will be concentrated in the mounting and in the implementation of the command in the *state machine*. All the necessary electric and electronic components have been already selected and ordered
- The variable *SA* (steering actuator) can have two values; 0 = “deactivated” and 1 = “activated”. Deactivated means no power supply to the actuator. Two ideas need to be tested for this implementation; either to use the CAN communication between the central control unit and the steering actuator controller, or implementing a similar relay-MOSFET scheme, as for *TS*

- The variable *strategy* inside the *missions* will be the trigger for the control strategies, which the team is preparing. So, the virtual integration between *strategies* and *state machine* and its physical test are other future steps of this work

The *Watchdog sequence* in the *EBS check-up* needs also to be studied and implemented.

The central control unit will be a *dSpace MicroAutoBox* already mounted in the vehicle. *dSpace Scalexio* will be used instead for the translation from Simulink environment to real control environment. This translation will cause some modifications in the *state machine* chart due to the integration with real signals and with the autonomous strategies for the missions.

In the appendix, bigger images of the models in Chapter 4 are reported. More precisely, the last overall system and all the last versions of the sub-systems can be better read.

References

[1] Perrino M. (2020), *Quante auto e moto circolano in Italia? ACI risponde*, Web link: https://www.motorbox.com/auto/magazine/vivere-auto/quante-auto-moto-circolanti-italia-2019-numeri-aci, verified in 07/09/2020

[2] History.com editors (2010), *Automobile History*, Web link: https://www.history.com/topics/inventions/automobiles, verified in 07/09/2020

[3] Wikipedia (2020), *Storia dell'automobile*, Web link: https://it.wikipedia.org/wiki/Storia_dell%27automobile, verified in 07/09/2020

[4] World Health Organization (2018), *The top 10 causes of death,* Web link: https://www.who.int/en/news-room/fact-sheets/detail/the-top-10-causes-of-death, verified in 07/09/2020

[5] Istituto Nazionale di Statistica (2020), *Incidenti stradali in Italia*, Web link: https://www.istat.it/it/archivio/245757, verified in 07/09/2020

[6] Morello L., L. R. Rossini, G. Pia and A. Tonoli (2011), *The Automotive Body Volume II: System Design*, Springer, pp. 41-42

[7] Szikora P. and N. Madarász (2017), *Self-driving cars – the human side*, working paper

[8] Anderson J. M., N. Kalra, K. D. Stanley, P. Sorensen, C. Samaras and O. A. Oluwatola (2016), *Autonomous Vehicle Technology: A guide for Policymakers*, RAND Corporation

[9] Wikipedia (2018), *DARPA Grand Challenge*, Web link: https://it.wikipedia.org/wiki/DARPA_Grand_Challenge, verified in 31/10/2020

[10] Herrmann A., W. Brenner and R. Stadler (2018), *Autonomous Driving: How the Driverless Revolution Will Change the World*, Emerald Group Pub Ltd, pp. 8-10

[11] Society of Automotive Engineers (2018), *SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles*, Web link: https://www.sae.org/news/press-room/2018/12/sae-

international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles, verified in 11/10/2020

[12] Wikipedia (2020), *Self-driving car*, Web link: https://en.wikipedia.org/wiki/Self-driving_car, verified in 12/10/2020

[13] Hylving L. and U. Schultze (2013), *Evolving the Modular Layered Architecture in Digital Innovation: The Case of the Car's Instrument Cluster,* conference paper, pp. 3-5

[14] Wikipedia (2020), *Remote Control Vehicle*, Web link: https://en.wikipedia.org/wiki/Remote_control_vehicle, verified in 02/11/2020

[15] Soo Z. and S. Dai (2019), *Remote control technology could be the key to moving autonomous driving forward*, Web link: https://www.scmp.com/tech/enterprises/article/3031585/why-remote-control-technology-could-be-key-moving-autonomous, verified in 03/11/2020

[16] *Remote operation of vehicles with 5G*, Web link: https://www.ericsson.com/en/mobility-report/articles/remote-monitoring-and-control-of-vehicles, verified in 03/11/2020

[17] Okumura B. and Prokhorov D. V. (2017), *Remote Operation of Autonomous Vehicle on Unexpected Environment*, patent application publication US 2017/0045885 A1

[18] C. A. Kemner and Peterson J. L. (1995), *Remote Control System and Method for an Autonomous Vehicle*, US patent n. 5 448 479

[19] *About Formula* Student, Web link: https://www.imeche.org/events/formula-student/about-formula-student, verified in 10/09/2020

[20] Kohavi Z. and N. K. Jha (2009), *Switching and Finite Automata Theory*, third edition, Cambridge University Press, pp. 265-272

[21] Wolf M. (2017), *Sequential machines*, Web link: https://www.sciencedirect.com/topics/computer-science/sequential-machine, verified in 14/09/2020

[22] Kinber E. and C. Smith (2001), *Theory of Computing: A Gentle Introduction*, Prentice Hall, pp. 13-23

[23] Università degli Studi Mediterranea di Reggio Calabria, *FSM: Macchine a Stati Finiti*, lecture notes

[24] Wikipedia (2020), *Mealy machine*, Web link: https://en.wikipedia.org/wiki/Mealy_machine, verified in 14/09/2020

[25] Wikipedia (2020), *Moore machine*, Web link: https://en.wikipedia.org/wiki/Moore_machine, verified in 14/09/2020

[26] Bucher R. (2013), *Introduzione a Stateflow*, lecture notes, Scuola Universitaria Professionale della Svizzera Italiana, Dipartimento di Tecnologie Innovative

[27] dSpace GmbH (2020), *Scalexio*, brochure, pp. 3-5 and pp. 93

[28] Wikipedia (2020), *Hardware in the loop*, Web link: https://it.wikipedia.org/wiki/Hardware-in-the-loop, verified in 07/09/2020

[29] Wikipedia (2020), *Rapid control prototyping*, Web link: https://en.wikipedia.org/wiki/Rapid_control_prototyping, verified in 07/09/2020

[30] dSpace GmbH (2020), *MicroAutoBox II*, brochure, pp. 2-3

[31] dSpace GmbH (2020), *Fullpassing*, Web link: https://www.dspace.com/en/pub/home/products/systems/functp/fullpassing.cfm, verified in 14/11/2020

[32] Formula Student (2019), *Formula Student Rules 2020*, version 1.0

[33] Formula Student Germany (2018), *FSG Competition Handbook 2019*, version 1.0

[34] Gross-Funk GmbH (2017), *Gross-Funk radio remote + automation*, instruction manual

[35] Wikipedia (2020), *CANopen*, Web link: https://en.wikipedia.org/wiki/CANopen#Device_model/home/products/systems/functp/fullpassing.cfm#143_1400, verified in 28/09/2020

[36] CiA (2002*)*, *CANopen: Application layer and communication profile*, version 4.0.2, pp.21-64 and pp.75-79

[37] CiA, *CANopen*, Web link: https://www.can-cia.org/canopen/, verified in 28/09/2020

[38] CiA, *Network management (NMT)*, Web link: https://www.can-cia.org/can-knowledge/canopen/network-management/, verified in 28/09/2020

[39] CiA, *Process data object (PDO)*, Web link: https://www.can-cia.org/can-knowledge/canopen/pdo-protocol/, verified in 28/09/2020

[40] Corrigan S. (2002), *Introduction to the Controller Area Network (CAN)*, application report revised in 2016, Texas Instruments

[41] Wikipedia (2020), *CAN bus*, Web link: https://en.wikipedia.org/wiki/CAN_bus, visited in 10/10/2020

[42] Leonardi E. (2019), *CAN bus*, lecture notes

[43] Hennessy B. (2019), *An introduction to J1939 and DBC files*, Web link: https://www.kvaser.com/developer-blog/an-introduction-j1939-and-dbc-files/#:~:text=The%20DBC%20file%20is%20an,data%20within%20the%20CAN%20frame, verified in 02/10/2020

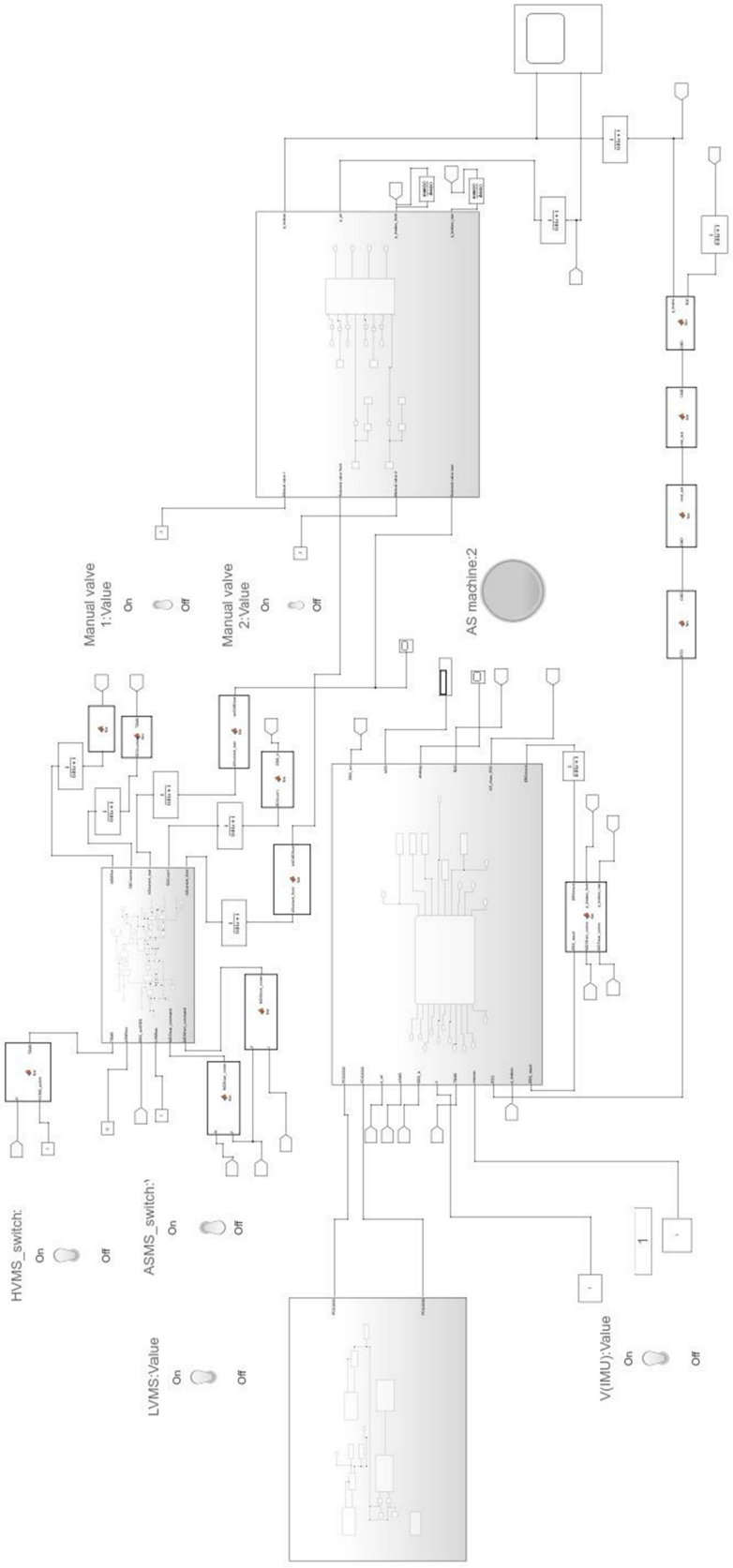
[44] Stollberger M. and M. Gebhardt (2018), *EBS Reference Guide 2019*, version 1.0, Formula Student Germany Driverless

[45] MathWorks, *Model Finite State Machines*, Web link: https://it.mathworks.com/help/stateflow/gs/finite-state-machines.html, verified in 15/09/2020

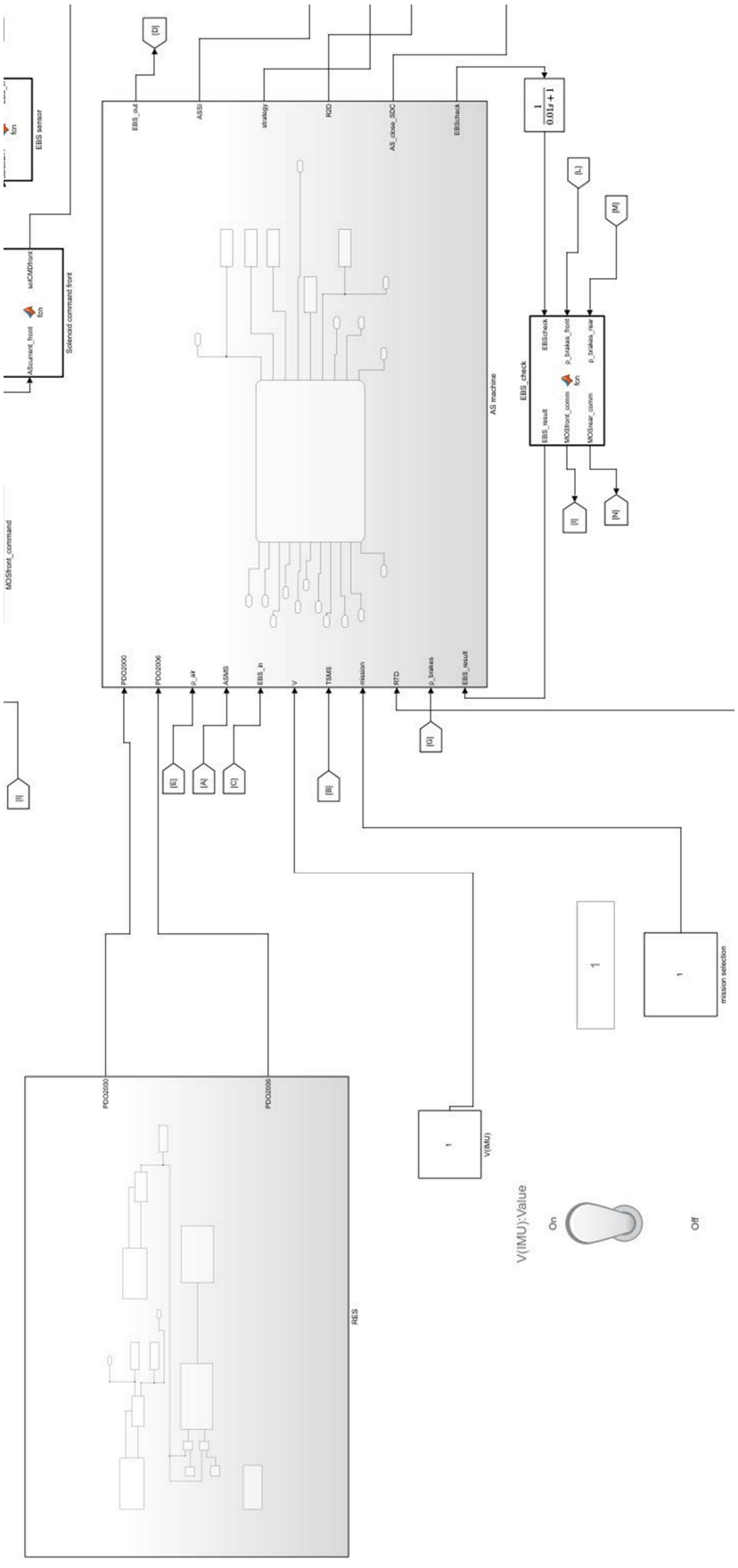
[46] MathWorks, *Syntax for states and transitions*, Web link: https://it.mathworks.com/help/stateflow/syntax-for-states-and-transitions.html?s_tid=CRUX_lftnav, verified in 20/10/2020

Appendix

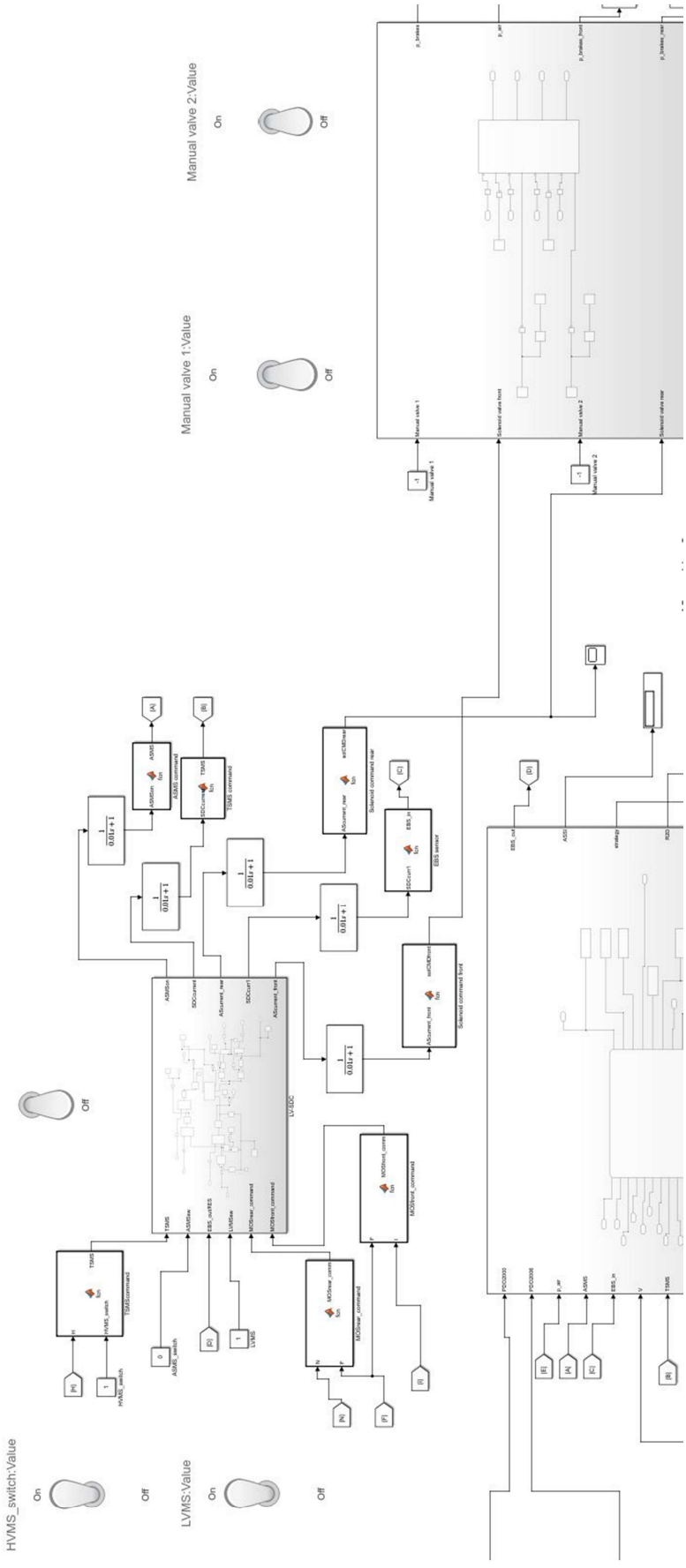
1. Complete model: overview



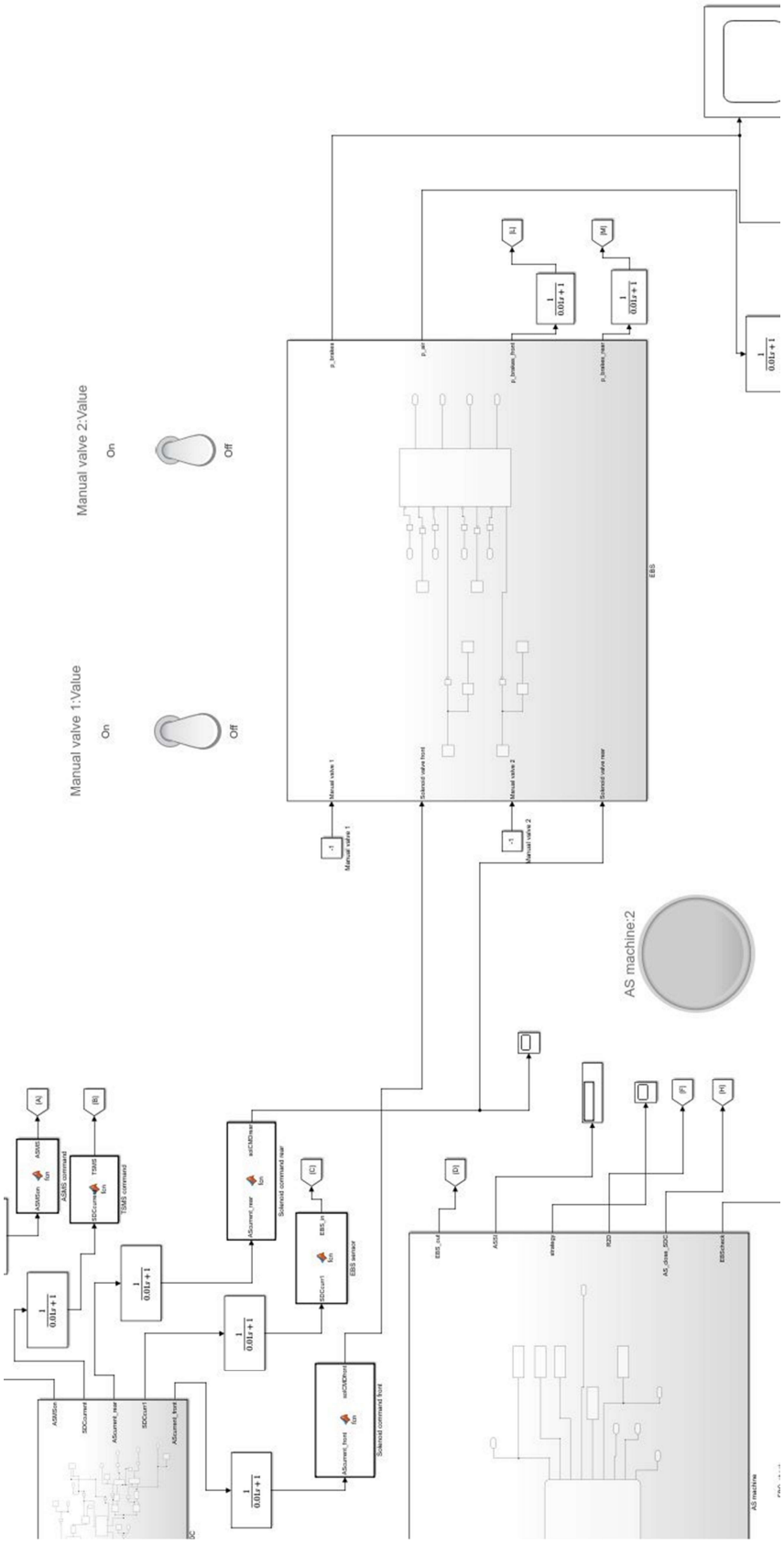
2. Complete model: RES – AS state machine link



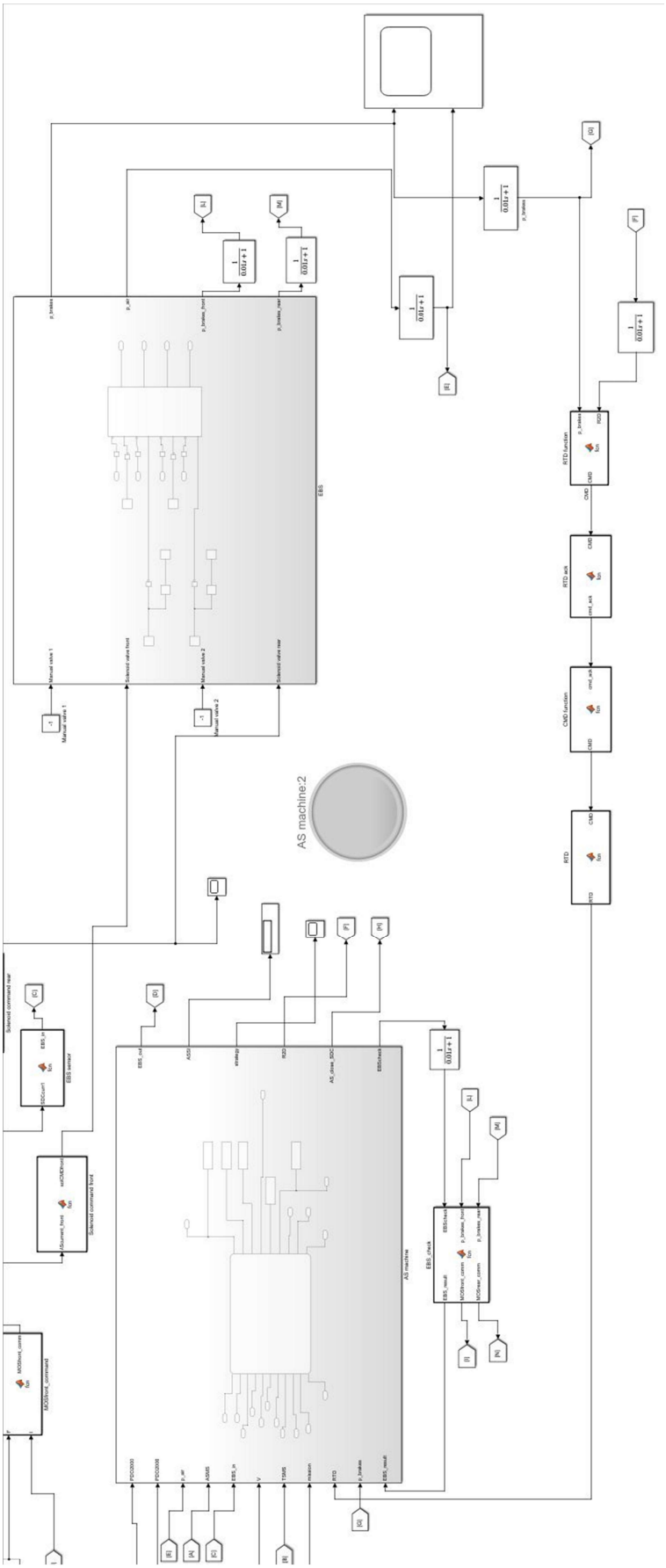
3. Complete model: Electrical scheme sub-system



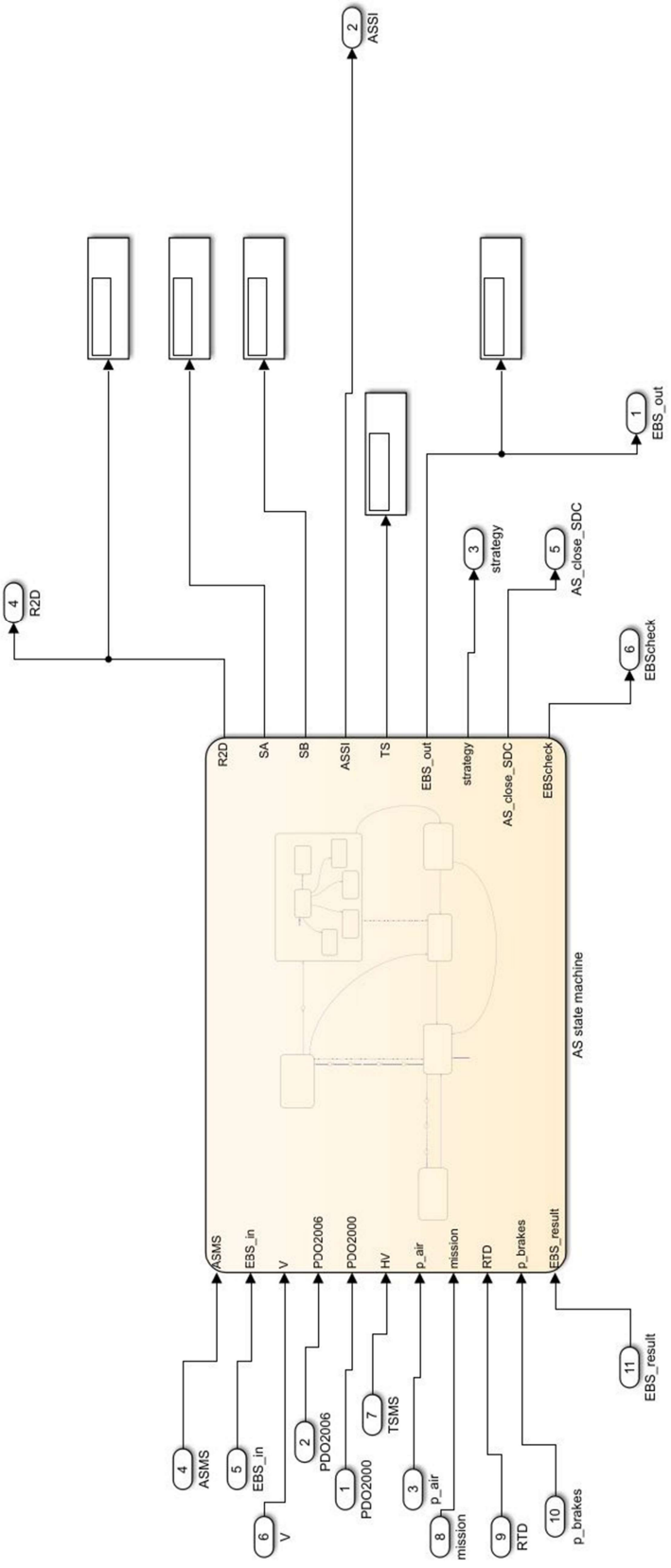
4. Complete model: EBS sub-system



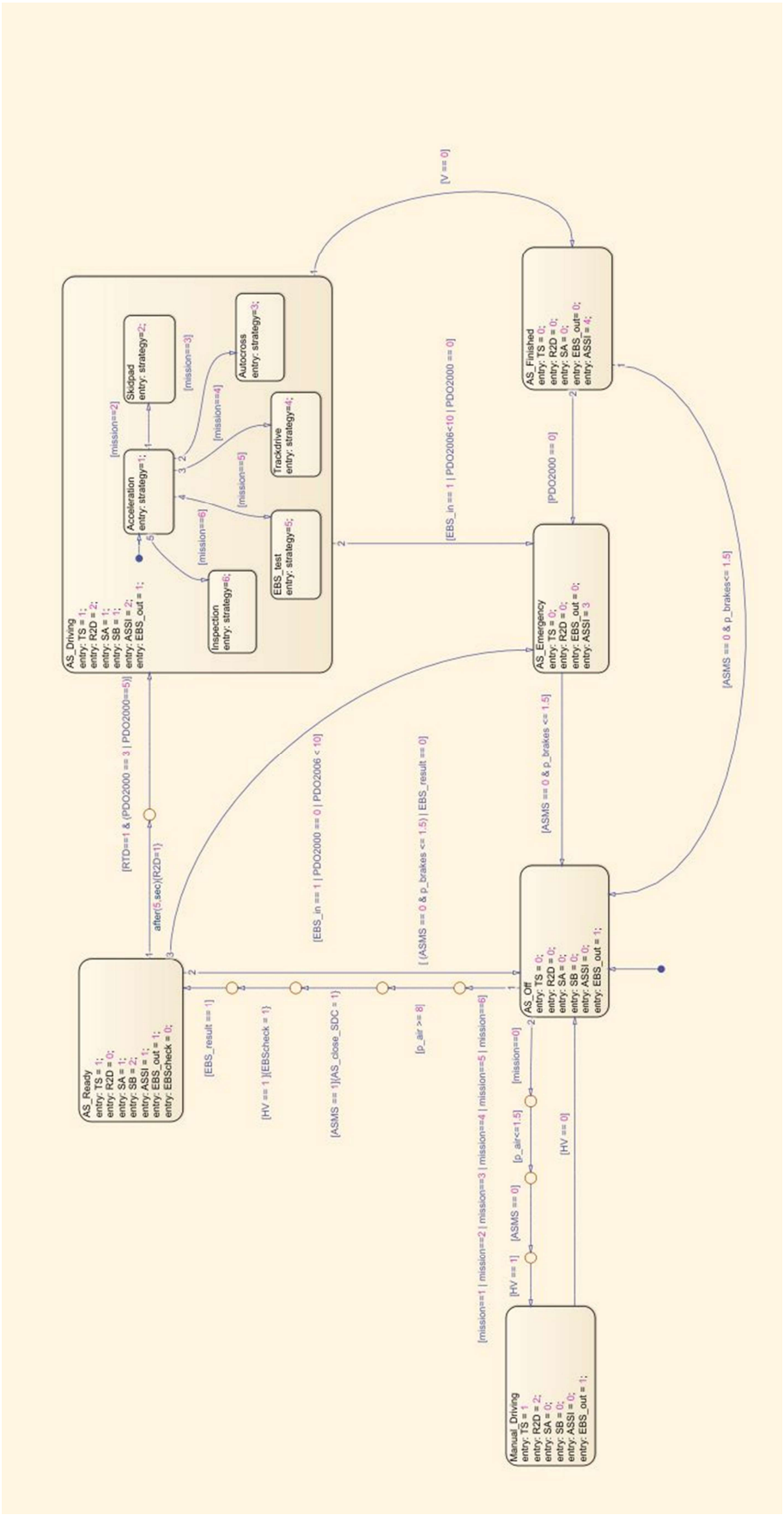
5. Complete model: EBS – AS state machine link



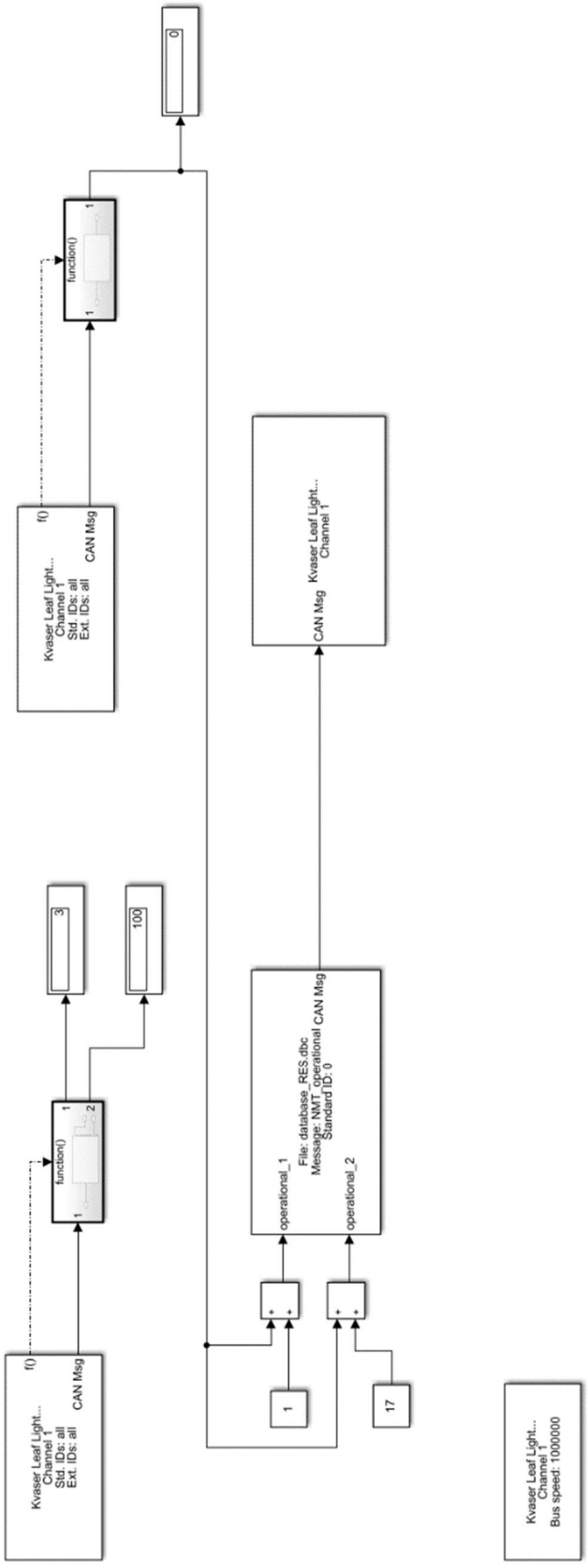
6. AS state machine sub-system



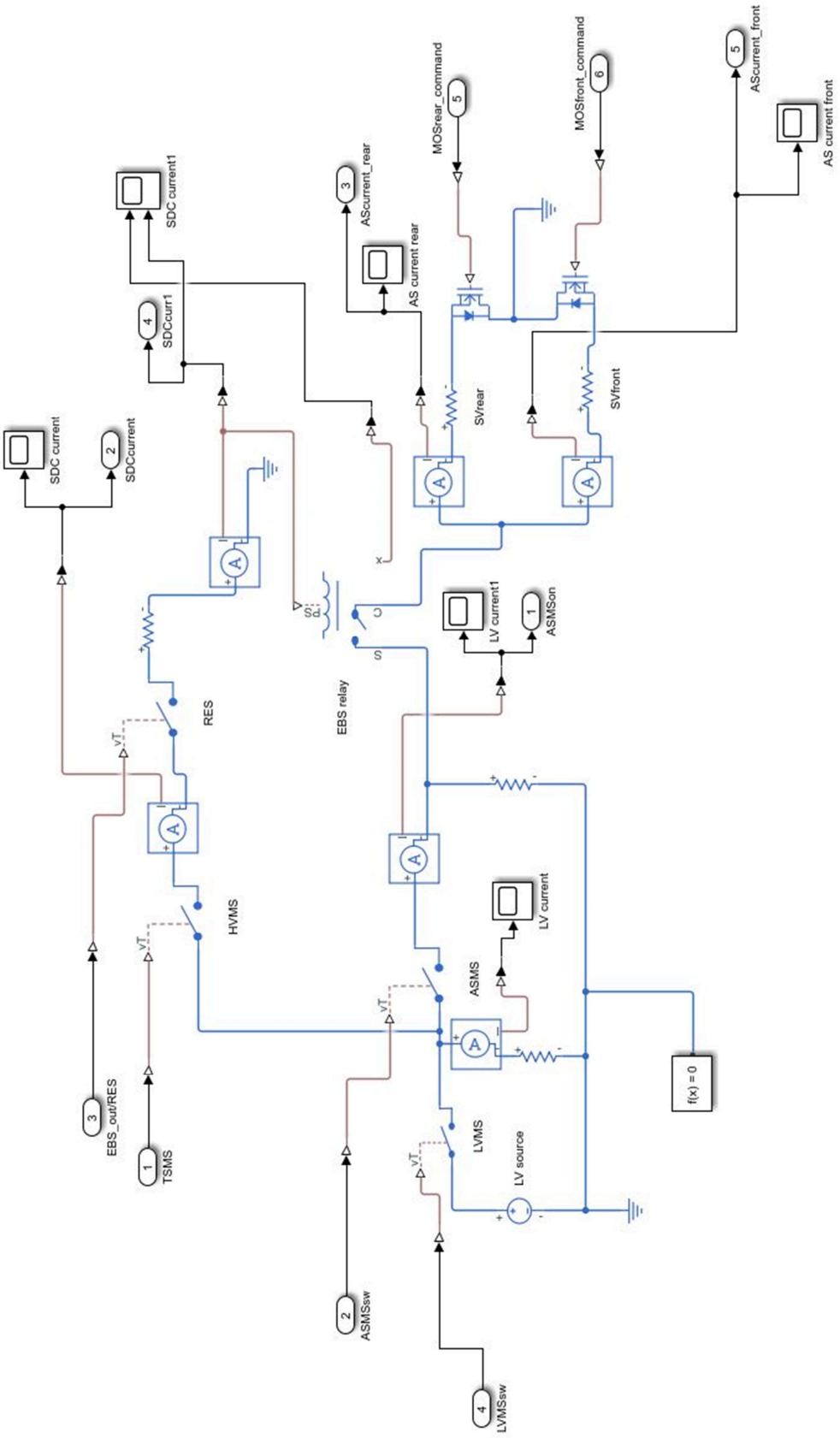
7. AS state machine final chart



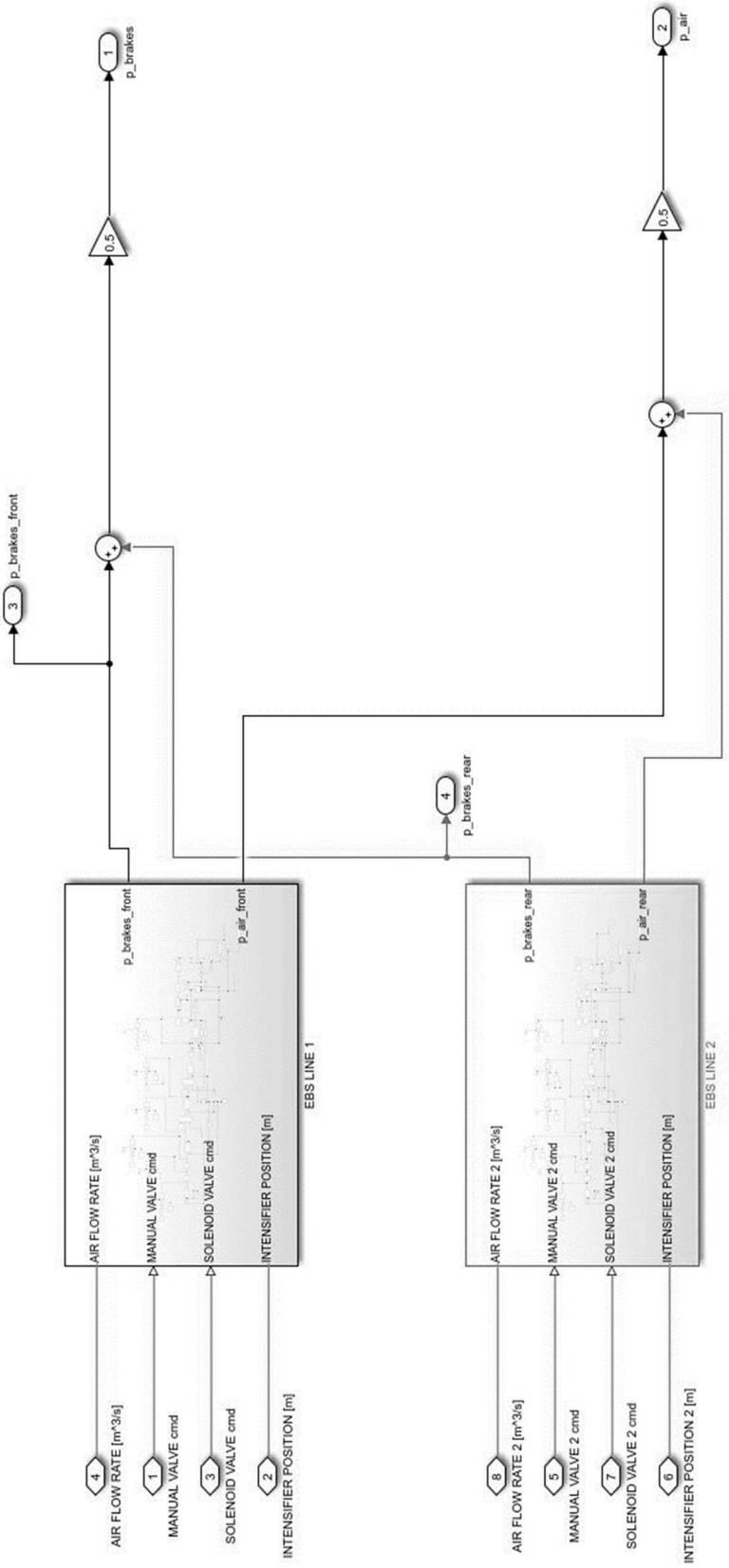
8. RES communication scheme



9. Electrical circuit



10. EBS front and rear sub-systems



11. EBS hydro-pneumatic model